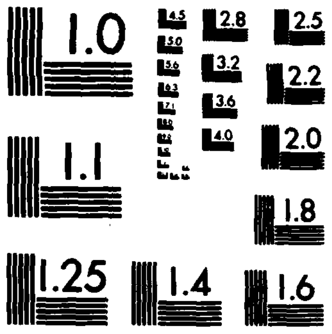

1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3

(12)

AD A 137436

RESEARCH ON INTERACTIVE ACQUISITION AND USE OF KNOWLEDGE

Final Report
Covering the Period July 3, 1980 to Nov. 30, 1983

November 1983

Principal Investigators:
Barbara J. Grosz, Program Director, Natural Language
Mark E. Stickel, Senior Computer Scientist

Artificial Intelligence Center
Computer Science and Technology Division

Prepared for:

Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, Virginia 22209

Attention: Commander Ronald B. Ohlander

SRI Project 1894

Preparation of this paper was supported by the Defense Advanced Research Projects Agency under Contract N00039-80-C-0575 with the Naval Electronic Systems Command.

SRI International
333 Ravenswood Avenue
Menlo Park, California 94025-3493
Telephone: (415) 326-6200
Cable: SRI INTL MPK
TWX: 910-373-2046
Telex: 334 486

83 12 20 154

1. Introduction

This report summarizes research done on the KLAUS project from July 1981 to July 1983.

In July 1979 we began work on design and implementation of the first in a series of KLAUS systems. This initial effort, conducted as a subtask of the LADDER project and continued under later funding, was intended primarily to demonstrate the feasibility of our ideas regarding knowledge acquisition. This work culminated in a proof-of-concept system called NANOKLAUS (operational in March 1980) which was based on a semantic grammar (derived from LIFER [42]) and a small natural-deduction theorem prover. The concepts underlying NANOKLAUS are described in the technical literature [37] and in our reports to DARPA.

In July 1980 we began work on the second KLAUS system, called MICROKLAUS, and on the technological base needed to support it. The MICROKLAUS demonstration system was constructed in 1981. The major components of this system differed significantly from those of NANOKLAUS. In particular, NANOKLAUS's simple language component was replaced by DIALOGIC [36], a sophisticated linguistic-analysis system based on the DIAGRAM grammar [110], and a nonclausal-resolution based theorem prover was developed to provide a major expansion of the underlying representation and deductive capabilities of the system. These more theoretically sound components were essential to moving beyond a proof-of-concept system to provide a base for potentially useful and truly extendible ones.

Our initial experience with MICROKLAUS led us to redesign the parsing and translation system to provide for a declarative semantics that is easier both to extend and to maintain, and to augment the deduction system along a number of dimensions. These development efforts proceeded independently, with the natural-language-processing and deduction components being only recently rejoined. We have also made significant progress on several fundamental problems of natural-language semantics and on specifying the planning and reasoning capabilitities [illegible] needed for generating adequate responses. Our systems have been moved from the DEC 2060 to the Symbolics 3600 LISP machine.

We expect these new components to provide the core of a new KLAUS system with significantly more powerful capabilities for communication in natural language. The base they provide also leaves us in a position to attack several fundamental research problems holding back the development of advanced KLAUS systems and other types of advanced systems that require high-quality natural-language capabilities.

The following outlines the contents of the remaining sections of this report.

1.1. MICROKLAUS Demonstration System

In Section 2 we present an annotated transcript of a dialogue with the MICROKLAUS demonstration system. The dialogue illustrates the natural-language processing capability of the system, including the capacity to deal with unknown words, in the context of an assertional-database application dealing with ships, commanders, etc. The types of assertions that can be made and questions that can be asked transcend those allowed for an ordinary database; some require nontrivial reasoning, such as reasoning by contradiction.

1.2. Natural-Language Processing

Section 3 describes the English translation system called PATR-I, our initial replacement of DIALOGIC/DIAGRAM by a more declarative translation facility. It presents a scheme for syntax-directed translation that mirrors compositional model-theoretic semantics and was used to specify a semantically interesting fragment of English, including such constructs as tense, aspect, modals, and various lexically controlled verb complement structures. PATR-I was embedded in a question-answering system that replied appropriately to questions requiring the computation of logical entailments.

The components of KLAUS that parse an input sentence and translate it into one or more expressions in logical form were developed further and implemented under the name

PATR-II. While radically different in design and function, PATR-II is a direct descendant of DIALOGIC/DIAGRAM and PATR-I. Most of the PATR-II research and development was done in the context of the KLAUS project, but this work has also benefited from basic research on metagrammatical formalisms, conducted under NSF grant number IST-8103550, that influenced several decisions and led to some of the notations employed for encoding linguistic generalizations.

After a period of relatively independent development of the formalism and initial test implementations, work on the current implementation was begun with the goal of integrating PATR-II with the other components of KLAUS. As a result, PATR-II has become an experimental, expandable, state-of-the-art parsing and translation component for the KLAUS system.

The name PATR-II was first used to designate a powerful formalism for grammar writing. The formalism was later equipped with a defined strategy of usage, together with a notation supporting such usage. Several experimental grammars were written in this format, one of which was selected for the latest KLAUS demonstration system. Section 4 describes the PATR-II system.

Section 5 discusses a parsing technique whose behavior resembles human behavior in the intepretation of certain problematical types of sentences. Native speakers of English show definite and consistent preferences for certain readings of syntactically ambiguous sentences. A user of a natural-language-processing system would naturally expect it to reflect the same preferences. Thus, such systems must model in some way the *linguistic performance* as well as the *linguistic competence* of the native speaker. We have developed a variant of the LALR(1) shift-reduce algorithm that models the preference behavior of native speakers for a range of syntactic-preference phenomena reported in the psycholinguistic literature, including the recent data on lexical preferences. The algorithm yields the preferred parse deterministically, without building multiple parse trees and choosing among them. As a side effect, it displays appropriate behavior in processing the much discussed garden-path sentences. The parsing algorithm has been implemented and has confirmed the feasibility of our approach to the modeling of these phenomena.

Section 6 discusses the necessity for a natural-language-processing system to be able to infer a user's domain goals as well as his communicative goals in order to be useful. Detailed examples of a hypothetical advice-giving program that is an expert on the MM mail system are presented illustrating how the system could infer the domain goals of the user and respond helpfully rather than literally to the user's questions.

1.3. Automated Deduction

Section 7 examines the role that formal logic ought to play in representing and reasoning with commonsense knowledge and is part of the rationale for the logic-based approach we employ for reasoning in KLAUS. We take issue with the commonly held view that the use of representations based on formal logic is inappropriate in most applications of artificial intelligence. We argue to the contrary that there is an important set of issues, involving incomplete knowledge of a problem situation, that so far have been addressed only by systems based on formal logic and deductive inference, and that, in some sense, probably can be dealt with only by systems based on logic and deduction. We further show that the experiments of the late 1960s on problem-solving by theorem-proving did not show that the use of logic and deduction in AI systems was necessarily inefficient, but rather that what was needed was better control of the deduction process, combined with more attention to the computational properties of axioms.

Section 8 describes the KLAUS deduction system. It consists of a theorem-proving program presently being developed and run on the Symbolics 3600 LISP machines. Earlier versions of the program running in INTERLISP on the DEC 2060 were used in the MICROKLAUS demonstration system and in the PATR-I assertional-database example. The current version of the program is used with the PATR-II natural-language-processing component on the 3600. The most important characteristics of the program are: nonclausal resolution is used as the inference system, eliminating some of the redundancy and unreadability of clause-based systems; a connection graph is used to represent permitted resolution operations, restricting the search space and facilitating the use of graph searching for efficient deduction; demodulation and special unification are used for building in equational theories; heuristic search and special logical connectives are used for program control.

In Section 9 we present the theory-resolution procedures for substantially increasing the power of automated-deduction systems by using more information in the basic matching operations. Theory resolution constitutes a set of complete procedures for building nonequational theories into a resolution theorem-proving program so that axioms of the theory need never be resolved upon. It is related to and extends the use of demodulation and special unification to build in equational theories. Total theory resolution uses a decision procedure that is capable of determining inconsistency of any set of clauses using predicates in the theory. Partial theory resolution employs a weaker decision procedure that can determine potential inconsistency of a pair of literals. Applications include the building in of both mathematical and special decision procedures, such as for the taxonomic information furnished by a knowledge representation system. Some partial theory resolution operations have been incorporated in the KLAUS deduction system.

Section 10 provides details on another approach to automated deduction. An extension of Prolog, based on the model elimination theorem-proving procedure, would permit production of a logically complete Prolog technology theorem prover capable of performing inference operations at a rate approaching that of Prolog itself. We have developed a prototype implementation based on code written for the deduction system described in Section 9. This represents a more "brute force" style of deduction than that discussed in Sections 9 and 10.

Two of the components developed for the KLAUS deduction system are special unification for associativity and/or commutativity and demodulation. These components were debugged and improved during the successful attempt to prove a difficult theorem by using the Knuth-Bendix method that relies heavily on special unification and demodulation. The proof of this theorem was posed as a challenge problem by W.W. Bledsoe in his 1977 article on non-resolution theorem proving and was solved by only one other theorem prover, using a different approach. The use of cancellation laws to simplify derived reductions was an important discovery made in this effort. Section 11 describes the proof of this theorem. The Knuth-Bendix method may serve as the basis for incorporation of additional equality reasoning capability in the KLAUS deduction system.

2. MICROKLAUS Demonstration System

This section was written by Mark Stickel and Stanley Rosenschein.

2.1. General Description of MICROKLAUS

The MICROKLAUS system has four principal components whose effects are visible to the user:

- The DIALOGIC natural-language-processing system
- A capability for dealing with unknown words
- The sort hierarchy
- The theorem prover.

The DIALOGIC natural-language-processing system is used to translate user-inputted English-language sentences into *logical form*. Logical form is a formal language that unambiguously represents the logical content of English sentences. (If the sentence is ambiguous, there will be more than one logical form.) Logical form is "close" to natural language. For example, it has quantifiers other than the conventional universal (ALL) and existential (SOME) quantifiers to conveniently express the semantic intent of English: e.g., a THE quantifier expressing uniqueness and a WH quantifier used to express "what." It also has special constructions for specifying sets of objects, for expressing comparatives and superlatives, and so on.

The system is intended to be instructable in new domains, and must therefore be capable of assimilating new vocabulary. In MICROKLAUS, this is done through conversation with the user. Previously unknown words can be introduced to MICROKLAUS simply by including them in a sentence inputted to MICROKLAUS. Upon encountering a new word, MICROKLAUS will attempt to determine its part of speech from the linguistic context in which it appears and will engage the user in further dialogue to elicit additional information about the word or concept, such as what type of object is named by a new noun or what type of object can be described by a new adjective.

The sort hierarchy is a data structure in MICROKLAUS that contains information used by both DIALOGIC and the theorem prover. The sort hierarchy is a taxonomy of objects and certain relationships among them. The sort hierarchy stores information on ***subset*** ("every man is a person"), ***spanning*** ("every person is a man, a woman, or a child"), and ***disjointness*** ("no man is a woman, no woman is a child").

The theorem prover is the system component that enables MICROKLAUS to assimilate facts entered by the user and to answer his queries. Sentences translated by DIALOGIC into logical form are further processed into first-order predicate calculus formulas which are stored (facts) or proved (queries). The theorem prover can be characterized approximately as a nonclausal resolution theorem prover with set-of-support and ordering restrictions and a depth-first search strategy. It is capable of doing nontrivial reasoning with negation (it can, in effect, do reasoning by contradiction unlike input-resolution theorem provers) and permits implications to be used selectively in either forward or backward reasoning.

2.2. Annotated Transcript of a MICROKLAUS Session

This section presents a transcript of an actual session with MICROKLAUS that displays the system's ability to acquire information about a new domain and reason with it. First, we present an abbreviated transcript highlighting the "flow" of the session. This is followed by a more complete, annotated transcript that explains the interactions in more detail. User input is shown in **boldface**, computer output in regular type, and commentary in ***italics***. (Some trace information has been omitted from the original transcript for the sake of brevity.)

1—**every carrier is a ship**
OK
2—**is every carrier a ship**
YES
3—**every adams-class ship has a length of 500 feet**
OK
4—**the fox is an adams-c ss carr'**
OK

```
5—the fox has a length of 500 feet
I ALREADY KNEW THAT
6—the hoel has a length of 600 feet
OK
7—what ships have a length of 500 feet
FOX
8—smith is the commander of the fox
OK
9—who does command the fox
I DON'T KNOW ANY
10—(assert.fact '(all x (+commander+ x)
                             (all y (and (+ship+ y) (*of x y))
                                    (*new-v-command x y))))
OK
11—who commands the fox
SMITH
12—every defective carrier will require a thorough overhaul
OK
13—every ship that requires an overhaul is in the log
OK
14—the fox is not in the log
OK
15—is the fox defective
NO
```

We now review the transcript item by item:

1—every carrier is a ship

The user introduces the words and concepts of carrier and ship and states a relationship between them.

```
[ASSERT (ALL +CARRIER+1 (+CARRIER+ +CARRIER+1)
             (SOME +SHIP+2 (+SHIP+ +SHIP+2)
                   (EQ +CARRIER+1 +SHIP+2]
```

MICROKLAUS accepts the statement and translates it into logical form. The logical form states that "for every carrier there is some ship such that the carrier is the ship."

```
TRYING TO REFUTE (NOT (IMPLY (+CARRIER+ (SK.1)) (+SHIP+ (SK.1))))
```

MICROKLAUS attempts to prove the statement from the information it already has. Since a refutation procedure is used rather than an affirmation procedure, an attempt to refute the statement that not every carrier is a ship is made, rather than a direct attempt to prove the statement that every carrier is a ship.

The MICROKLAUS theorem prover is a theorem prover for the first order predicate calculus. Logical forms are simplified, translated into first-order predicate calculus, and skolemized (have existentially quantified variables replaced by Skolem functions and all quantifiers removed) in preparation for processing by the MICROKLAUS theorem prover.

In the above instance, the theorem prover is asked to refute the assertion that there is some carrier (referred to by the Skolem constant SK.1) that is not a ship.

ASSERTING (IMPLY (+CARRIER+ +CARRIER+1) (+SHIP+ +CARRIER+1))

Having failed to prove that every carrier is a ship from the information it already had, MICROKLAUS adds that fact.

Since ship is a new concept to MICROKLAUS, MICROKLAUS initiates the following dialogue with the user to determine what kind of object a ship is and place it correctly in the sort hierarchy. MICROKLAUS doesn't ask about carriers because it already knows what it needs about carriers from the statement that "every carrier is a ship."

What is a SHIP? **physical% object**
You're saying that anything that is a SHIP is also a PHYSICAL OBJECT.
Is LIVING CREATURE a proper subclass of SHIP? **no**
Is LIVING CREATURE necessarily composed of entirely different members from SHIP? **yes**
Do LIVING CREATURE and SHIP span the set of all PHYSICAL OBJECTS? **no**

In the dialogue, the user is asked for information on (1) where is ship in the sort hierarchy (it is told that ship is a physical object), (2) whether being a ship is distinct from being something else that is also a physical object, and (3) whether ship, in combination with other types of physical objects, covers the complete range of all possible physical objects.

Although some of this information is used in DIALOGIC, at the present time, the MICROKLAUS theorem prover uses only (1). An assertion is made to the theorem prover that every ship is a physical object:

ASSERTING (IMPLY (+SHIP+ X) (+PHYSICAL OBJECT+ X))
Ok, now I understand SHIP.

2—**is every carrier a ship**

That MICROKLAUS understands the previous statement that "every carrier is a ship" can be verified by asking "is every carrier a ship." The question is translated into logical form:

```
[QUERY (ALL +CARRIER+1 (+CARRIER+ +CARRIER+1)
           (SOME +SHIP+2 (+SHIP+ +SHIP+2)
                 (EQ +CARRIER+1 +SHIP+2]
```

The logical form is translated into a refutation attempt that succeeds. The response to the question is "YES":

```
TRYING TO REFUTE (NOT (IMPLY (+CARRIER+ (SK.2)) (+SHIP+ (SK.2))))
*** DONE ***
YES
```

3—**every adams-class ship has a length of 500 feet**

An assertion about Adams-class ships is made. Both Adams-class and length are previously unknown words to MICROKLAUS.

The sentence is translated into logical form:

```
[ASSERT (ALL +SHIP+1 (AND (+SHIP+ +SHIP+1)
                          (*NEW-ADJ-ADAMS-CLASS +SHIP+1))
     (SOME +LENGTH+2 (AND (+LENGTH+ +LENGTH+2)
                          (*OF +LENGTH+2 (+FOOT+ 500)))
        (*HAVE +SHIP+1 +LENGTH+2]
```

An attempt is made to prove the statement:

```
TRYING TO REFUTE (NOT (IMPLY (AND (+SHIP+ (SK.3)) (*NEW-ADJ-ADAMS-CLASS (SK.3)))
                             (AND (+LENGTH+ +LENGTH+2)
                                  (*OF +LENGTH+2 (+FOOT+ 500))
                                  (*HAVE (SK.3) +LENGTH+2))))
```

The statement is asserted after the proof attempt fails:

```
ASSERTING (IMPLY (AND (+SHIP+ +SHIP+1) (*NEW-ADJ-ADAMS-CLASS +SHIP+1))
                 (AND (+LENGTH+ (SK.4 +SHIP+1))
                      (*OF (SK.4 +SHIP+1) (+FOOT+ 500))
                      (*HAVE +SHIP+1 (SK.4 +SHIP+1))))
```

MICROKLAUS then asks the user to further define the concept length. His responses of "physical% object" to the query about ships and "linear-measure" to this query about length refer to already existing entries in the sort hierarchy.

What is a LENGTH? **linear-measure**
You're saying that anything that is a LENGTH is also a LINEAR-MEASURE.

ASSERTING (IMPLY (+LENGTH+ X) (+LINEAR-MEASURE+ X))
Ok, now I understand LENGTH.

MICROKLAUS then asks what is the most general class of objects to which the adjective "Adams-class" can be applied. This information is used in the parsing of later sentences in which the word "Adams-class" appears.

What is the most general class of thing which can be ADAMS-CLASS ? **ship**

4—**the fox is an adams-class carrier**

The Fox is identified as an Adams-class carrier. In this linguistic context, "the Fox" could refer either to a single object or a class of objects. MICROKLAUS asks which interpretation is intended:

Is there just one thing called FOX (1)

```
or are you referring to one particular FOX of many (2) ? 1
```

The sentence is translated into logical form, an attempt to prove the statement is made and fails, and the assertions are made that Fox is Adams-class and Fox is a carrier:

```
(ASSERT (SOME +CARRIER+1 (AND (+CARRIER+ +CARRIER+1)
                              (*NEW-ADJ-ADAMS-CLASS +CARRIER+1))
        (EQ +FOX+ +CARRIER+1)))

TRYING TO REFUTE (OR (NOT (+CARRIER+ (+FOX+)))
                     (NOT (*NEW-ADJ-ADAMS-CLASS (+FOX+))))

ASSERTING (+CARRIER+ (+FOX+))

ASSERTING (*NEW-ADJ-ADAMS-CLASS (+FOX+))
```

5—**the fox has a length of 500 feet**

The user states that the Fox has a length of 500 feet. This fact is deducible from the statements that Fox is an Adams-class carrier, that every carrier is a ship, and that every Adams-class ship has a length of 500 feet. The attempt to verify that the statement was already known succeeds, and MICROKLAUS responds "I ALREADY KNEW THAT":

```
(ASSERT (SOME +LENGTH+1 (AND (+LENGTH+ +LENGTH+1)
                             (*OF +LENGTH+1 (+FOOT+ 500)))
              (*HAVE +FOX+ +LENGTH+1)))

TRYING TO REFUTE (NOT (AND (+LENGTH+ +LENGTH+1)
                           (*OF +LENGTH+1 (+FOOT+ 500))
                           (*HAVE (+FOX+) +LENGTH+1)))
*** DONE ***
I ALREADY KNEW THAT
```

6—**the hoel has a length of 600 feet**

Another ship and its length are introduced by the user.

```
Is there just one thing called HOEL (1)
or are you referring to one particular HOEL of many (2) ? 1

(ASSERT (SOME +LENGTH+1 (AND (+LENGTH+ +LENGTH+1)
                             (*OF +LENGTH+1 (+FOOT+ 600)))
         (*HAVE +HOEL+ +LENGTH+1)))

TRYING TO REFUTE (NOT (AND (+LENGTH+ +LENGTH+1)
                           (*OF +LENGTH+1 (+FOOT+ 600))
                           (*HAVE (+HOEL+) +LENGTH+1)))

ASSERTING (+LENGTH+ (SK.5))

ASSERTING (*OF (SK.5) (+FOOT+ 600))

ASSERTING (*HAVE (+HOEL+) (SK.5))
What is HOEL an instance of? ship
```

```
You're saying that HOEL is one instance of a SHIP.

ASSERTING (+SHIP+ (+HOEL+))
Ok, now I know about HOEL.
```

7—what ships have a length of 500 feet

> *The users asks what ships have a length of 500 feet. An attempt is made to prove that some ship has a length of 500 feet. Every ship for which it can be proved that it has a length of 500 feet is returned in the response to the query.*

```
[QUERY (WH +SHIP+1 (+SHIP+ +SHIP+1)
           (SOME +LENGTH+2 (AND (+LENGTH+ +LENGTH+2)
                                (*OF +LENGTH+2 (+FOOT+ 500)))
                (*HAVE +SHIP+1 +LENGTH+2]

TRYING TO REFUTE (NOT (AND (+SHIP+ +SHIP+1)
                           (+LENGTH+ +LENGTH+2)
                           (*OF +LENGTH+2 (+FOOT+ 500))
                           (*HAVE +SHIP+1 +LENGTH+2)
                           (ANSWER +SHIP+1)))
*** (ANSWER (+FOX+)) ***
FOX.
```

> *The Fox is the only ship MICROKLAUS knows about that has a length of 500 feet.*

8—smith is the commander of the fox

> *The user states that "Smith is the commander of the Fox." "Smith" and "commander" are both previously unknown words to MICROKLAUS. MICROKLAUS asks the user whether "the commander" is a single object or one of a class of objects:*

```
Is there just one thing called COMMANDER (1)
or are you referring to one particular COMMANDER of many (2) ? 2
```

> *MICROKLAUS translates the statement into logical form, attempts to prove it from the information it already has, fails, and asserts it:*

```
(ASSERT (THE +COMMANDER+1 (AND (+COMMANDER+ +COMMANDER+1)
                                (*OF +COMMANDER+1 +FOX+))
        (EQ +SMITH+ +COMMANDER+1)))

TRYING TO REFUTE (OR (NOT (+COMMANDER+ (+SMITH+)))
                     (NOT (*OF (+SMITH+) (+FOX+))))

ASSERTING (+COMMANDER+ (+SMITH+))

ASSERTING (*OF (+SMITH+) (+FOX+))
```

> *MICROKLAUS then converses with the user to place commander in the sort hierarchy:*

What is a COMMANDER? **person**
You're saying that anyone who is a COMMANDER is also a PERSON.
Which of the following, if any, are proper subclasses of COMMANDER: MAN, WOMAN, or CHILD? **none**
Which of the following classes, if any, could never have any members in common with COMMANDER: MAN, WOMAN, or CHILD ? **child**

What combination of the following subclasses, if any, together with COMMANDER, span the class of all PEOPLE (with or without overlapping): MAN, WOMAN, or CHILD ? **none**

ASSERTING (IMPLY (+COMMANDER+ X) (+PERSON+ X))
Ok, now I understand COMMANDER.

9—who does command the fox

> *The user asks MICROKLAUS who commands the Fox. "Command" is a previously unknown word to MICROKLAUS. It is deliberately introduced in the form "does command" in this sentence, so that its first appearance will be without any suffix (such as "s" in "commands"). In the future, MICROKLAUS will handle unknown words even when their first appearance is with a suffix.*

```
(QUERY (WH +PERSON+1 (+PERSON+ +PERSON+1)
                     (*NEW-V-COMMAND +PERSON+1 +FOX+)))

TRYING TO REFUTE (NOT (AND (+PERSON+ +PERSON+1)
                           (*NEW-V-COMMAND +PERSON+1 (+FOX+))
                           (ANSWER +PERSON+1)))
I DON'T KNOW ANY
```

> *MICROKLAUS doesn't know any person who commands the Fox. The reason is that, although it was asserted that Smith is the commander of the Fox, no relationship between "commander" and "command" has been established. The assertion "the commander of a ship commands the ship" is made directly to the MICROKLAUS theorem prover:*

```
10—(assert.fact '(all x (+commander+ x)
                        (all y (and (+ship+ y) (*of x y))
                               (*new-v-command x y))))
OK

ASSERTING (IMPLY (+COMMANDER+ X)
                 (IMPLY (AND (+SHIP+ Y) (*OF X Y)) (*NEW-V-COMMAND X Y)))
```

> *In the future, the DIALOGIC language system will handle anaphoric sentences such as "the commander of a ship commands the ship" and this direct assertion to the theorem prover will be unnecessary.*

11—who commands the fox

> *The user again asks who commands the Fox. Now that the word command is known to MICROKLAUS, it can appear with a suffix in "commands."*

```
(QUERY (WH +PERSON+1 (+PERSON+ +PERSON+1) (*NEW-V-COMMAND
+PERSON+1 +FOX+)))
TRYING TO REFUTE (NOT (AND (+PERSON+ +PERSON+1)
                           (*NEW-V-COMMAND +PERSON+1 (+FOX+))
                           (ANSWER +PERSON+1)))
*** (ANSWER (+SMITH+)) ***
SMITH.
```

This time, MICROKLAUS succeeds in identifying Smith as the person who commands the Fox.

12—**every defective carrier will require a thorough overhaul**

The user makes another statement to MICROKLAUS. This statement is particularly interesting because of its high number of unknown words. The words "defective", "require", "thorough", and "overhaul" are all previously unknown to MICROKLAUS and their correct part of speech must all be determined from the sentence structure.

The sentence is translated into logical form, a proof attempt of it from previous information fails, and the statement is asserted:

```
[ASSERT (ALL +CARRIER+1 (AND (+CARRIER+ +CARRIER+1)
                             (*NEW-ADJ-DEFECTIVE  +CARRIER+1))
        (SOME +OVERHAUL+2 (AND (+OVERHAUL+ +OVERHAUL+2)
                               (*NEW-ADJ-THOROUGH  +OVERHAUL+2))

            (*NEW-V-REQUIRE  +CARRIER+1  +OVERHAUL+2]
TRYING TO REFUTE (NOT (IMPLY (AND (+CARRIER+ (SK.6))
                                  (*NEW-ADJ-DEFECTIVE (SK.6)))
                             (AND (+OVERHAUL+ +OVERHAUL+2)
                                  (*NEW-ADJ-THOROUGH +OVERHAUL+2)
                                  (*NEW-V-REQUIRE (SK.6) +OVERHAUL+2))))
ASSERTING (IMPLY (AND (+CARRIER+ +CARRIER+1)
                      (*NEW-ADJ-DEFECTIVE +CARRIER+1))
                 (AND (+OVERHAUL+ (SK.7 +CARRIER+1))
                      (*NEW-ADJ-THOROUGH (SK.7 +CARRIER+1))
                      (*NEW-V-REQUIRE +CARRIER+1 (SK.7 +CARRIER+1))))
```

MICROKLAUS then asks what kind of object an overhaul is, and also asks what kinds of objects the adjectives "defective" and "thorough" can modify:

What is an OVERHAUL? **abstract% object**
You're saying that anything that is an OVERHAUL is also an ABSTRACT OBJECT.
Which of the following, if any, are proper subclasses of OVERHAUL: NUMBER, MEASURE, UNIT-OF-MEASURE, or LEGAL ABSTRACTION? **none**
Which of the following classes, if any, could never have any members in common with OVERHAUL: NUMBER, MEASURE, UNIT-OF-MEASURE, or LEGAL ABSTRACTION ? **all**

What combination of the following subclasses, if any, together with OVERHAUL, span the class of all ABSTRACT OBJECTS (with or without overlapping): NUMBER, MEASURE, UNIT-OF-MEASURE, or LEGAL ABSTRACTION ? **none**

```
ASSERTING (IMPLY (+OVERHAUL+ X) (+ABSTRACT OBJECT+ X))
Ok, now I understand OVERHAUL.
What is the most general class of thing which can be THOROUGH ? thing
What is the most general class of thing which can be DEFECTIVE ? thing
```

13—every ship that requires an overhaul is in the log

Is there just one thing called LOG (1)
or are you referring to one particular LOG of many (2) ? **1**

```
(ASSERT (ALL +SHIP+1 (SOME +OVERHAUL+2 (+OVERHAUL+ +OVERHAUL+2)
                           (AND (+SHIP+ +SHIP+1)
                                (*NEW-V-REQUIRE +SHIP+1 +OVERHAUL+2)))

          (*IN +SHIP+1 +LOG+)))

TRYING TO REFUTE (NOT (IMPLY (AND (+OVERHAUL+ (SK.9))
                                  (+SHIP+ (SK.8))
                                  (*NEW-V-REQUIRE (SK.8) (SK.9)))
                             (*IN (SK.8) (+LOG+))))

ASSERTING (IMPLY (AND (+OVERHAUL+ +OVERHAUL+2)
                      (+SHIP+ +SHIP+1)
                      (*NEW-V-REQUIRE +SHIP+1 +OVERHAUL+2))
                 (*IN +SHIP+1 (+LOG+)))
```

What is LOG an instance of? **physical% object**
You're saying that LOG is one instance of a PHYSICAL OBJECT.

```
ASSERTING (+PHYSICAL OBJECT+ (+LOG+))
Ok, now I know about LOG.
```

14—the fox is not in the log

```
(ASSERT (NOT (*IN +FOX+ +LOG+)))

TRYING TO REFUTE (*IN (+FOX+) (+LOG+))

ASSERTING (NOT (*IN (+FOX+) (+LOG+)))
```

15—is the fox defective

In the context of the preceding information given MICROKLAUS, the user asks whether the Fox is defective.

```
(QUERY (*NEW-ADJ-DEFECTIVE +FOX+))
```

MICROKLAUS attempts to prove that the Fox is defective (by refuting the statement that it is not defective):

```
TRYING TO REFUTE (NOT (*NEW-ADJ-DEFECTIVE (+FOX+)))
```

This proof attempt fails. So, MICROKLAUS tries to prove its negation, i.e., tries to prove that the Fox is not defective. (This approach is more secure than just assuming that the Fox is not defective on the basis of failing to find a proof that it is.)

```
TRYING TO REFUTE (*NEW-ADJ-DEFECTIVE (+FOX+))
*** DONE ***
NO
```

The attempt to prove that the Fox is not defective succeeds, and hence the answer to the question "Is the Fox defective" is "NO."

This proof requires a form of reasoning by contradiction since the statement that the Fox is not defective must be derived from the contradiction between the Fox possibly being defective (implying its needing an overhaul and thus being in the log) and the fact that the Fox is not in the log.

3. Translating English into Logical Form

This section was written by Stanley Rosenschein and Stuart Shieber.

3.1. Introduction

When contemporary linguists and philosophers speak of "semantics," they usually mean model-theoretic semantics—mathematical devices for associating truth conditions with sentences. Computational linguists, on the other hand, often use the term "semantics" to denote a phase of processing in which a data structure (e.g., a formula or network) is constructed to represent the meaning of a sentence and serve as input to later phases of processing. (A better name for this process might be "translation" or "transduction.") Whether one takes "semantics" to be about model theory or translation, the fact remains that natural languages are marked by a wealth of complex constructions—such as tense, aspect, moods, plurals, modality, adverbials, degree terms, and sentential complements—that make semantic specification a complex and challenging endeavor.

Computer scientists faced with the problem of managing software complexity have developed strict design disciplines in their programming methodologies. One might speculate that a similar requirement for manageability has led linguists (since Montague, at least) to follow a discipline of strict compositionality in semantic specification, even though model-theoretic semantics *per se* does not demand it. Compositionality requires that the meaning of a phrase be a function of the meanings of its immediate constituents, a property that allows the grammar writer to correlate syntax and semantics on a rule-by-rule basis and keep the specification modular. Clearly, the natural analogue to compositionality in the case of translation is *syntax-directed translation;* it is this analogy that we seek to exploit.

We describe a syntax-directed translation scheme that bears a close resemblance to model-theoretic approaches and achieves a level of perspicuity suitable for the development of large and complex grammars by using a declarative format for specifying grammar rules. In our formalism, translation types are associated with the phrasal categories of English in much the way that logical-denotation types are associated with phrasal categories in model-theoretic semantics. The translation types are classes of *data objects* rather than abstract *denotations*, yet they play much the same role in the translation process that denotation types play in formal semantics.

In addition to this parallel between logical types and translation types, we have intentionally designed the language in which translation rules are stated to emphasize parallels between the syntax-directed translation and corresponding model-theoretic interpretation rules found in, say, the GPSG literature [30]. In the GPSG approach, each syntax rule has an associated semantic rule (typically involving functional application) that specifies how to compose the meaning of a phrase from the meanings of its constituents. In an analogous fashion, we provide for the translation of a phrase to be synthesized from the translations of its immediate constituents according to a local rule, typically involving *symbolic application* and λ-conversion.

It should be noted in passing that doing translation, rather than model-theoretic interpretation, offers the temptation to abuse the formalism by having the "meaning" (translation) of a phrase depend on *syntactic* properties of the translations of its constituents—for instance, on the order of conjuncts in a logical expression. There are several points to be made in this regard. First, without severe *a priori* restrictions on what kinds of objects can *be* translations (coupled with the associated strong theoretical claims that such restrictions would embody), it seems impossible to prevent such abuses. Second, as in the case of programming languages, it is reasonable to assume that there would emerge a set of stylistic practices that, for reasons of manageability and esthetics, would govern the actual form of grammars. Third, it is still an open question whether the model-theoretic program of strong compositionality will actually succeed. Indeed, whether it succeeds or not is of little concern to the computational linguist, whose systems, in any event, have no direct way of using the sort of abstract model being proposed and must generally be based on deduction (and hence translation).

The rest of this section discusses our work in more detail. Section 3.2 presents the grammar formalism and describes PATR-I, an implemented parsing and translation system that can accept a grammar in our formalism and use it to process sentences. Examples of the system's operation, including its application in a simple deductive question-answering system, are found in Section 3.3. Section 3.4 describes further extensions of the formalism and the parsing system. Section 3.5 contains sample grammar rules, meaning postulates (axioms) used by the question-answering system, a sample dialogue session.

3.2. A Grammar Formalism

3.2.1 General Characterization

Our grammar formalism is best characterized as a specialized type of augmented context-free grammar. That is, we take a grammar to be a set of context-free rules that define a language and, in the usual way, associate structural descriptions (parse trees) for each sentence in that language. Nodes in the parse tree are assumed to have a set of features that may assume binary values (*True* or *False*), and there is a distinguished attribute—the "translation"— whose values range over a potentially infinite set of objects, i.e., the translations of English phrases.

Viewed more abstractly, we regard translation as a binary relation between word sequences and logical formulas. The use of a relation is intended to incorporate the fact that many word sequences have several logical forms, while some have none at all. Furthermore, we view this relation as being composed (in the mathematical sense) of four simpler relations corresponding to the conceptual phases of analysis: (1) LEX (lexical analysis), (2) PARSE (parsing), (3) ANNOTATE (assignment of attribute values, syntactic filtering), and (4) TRANSLATE (translation proper, i.e., synthesis of logical form).

The domains and ranges of these relations are as follows:

Word Sequences $-LEX\rightarrow$
    Morpheme Sequences $-PARSE\rightarrow$
        Phrase Structure Trees $-ANNOTATE\rightarrow$
            Annotated Trees $-TRANSLATE\rightarrow$
                Logical Form

```
RULES:
   Constant COMP' = (λ P (λ Q (λ X (P (Q X)))))
   S → NP VP
     Trans:  VP'[NP']
   VP → TENSE V
     Anno:  [ ¬Transitive(V) ]
     Trans: { COMP'[TENSE'][V'] }

LEXICON:
   NP → John
     Anno:  [ Proper(NP) ]
     Trans: { John }
   TENSE → &past
     Trans: { (λ X (past X)) }
   V → go
     Anno:  [ ¬Transitive(V) ]
     Trans: { (λ X (go X)) }
```

Figure 1: Sample specification of augmented phrase structure grammar

The relational composition of these four relations is the full translation relation associating word sequences with logical forms. The subphases too are viewed as *relations* to reflect the inherent nondeterminism of each stage of the process. For example, the sentence "a hat by every designer sent from Paris was felt" is easily seen to be nondeterministic in LEX ("felt"), PARSE (postnominal modifier attachment), and TRANSLATE (quantifier scoping).

It should be emphasized that the correspondence between *processing* phases and these *conceptual* phases is loose. The goal of the separation is to make specification of the process perspicous and to allow simple, clean implementations. An actual system could achieve the net effect of the various stages in many ways, and numerous optimizations could be envisioned that would have the effect of folding back later phases to increase efficiency.

3.2.2 The Relations LEX, PARSE, and ANNOTATE

We now describe a characteristic form of specification appropriate to each phase, and illustrate how the word sequence "John went" is analyzed by stages as standing in the translation relation to "(past (go john))" according to the (trivial) grammar presented in Figure 1.

Lexical analysis is specified by giving a kernel relation between individual words and morpheme sequences[1] (or equivalently, a *mapping* from words to sets of morpheme sequences), for example:

```
John → (john);
went → (&past go);
persuaded → (&past persuade),
                (&ppl persuade);
```

The kernel relation is extended in a standard fashion to the full LEX relation. For example, "went" is mapped to the single morpheme sequence (&past go), and "John" is mapped to (john). Thus, by extension, "John went" is transformed to (John &past go) by the lexical analysis phase.

Parsing is specified in the usual manner by a context-free grammar. Through utilization of the context-free rules presented in the sample system specification shown in Figure 1, (John &past go) is transformed into the parse tree

```
(S (NP john)
   (VP (TENSE &past)
       (V go)))        .
```

Every node in the parse tree has a set of associated features. The purpose of ANNOTATE is to relate the bare parse tree to one that has been enhanced with attribute values, filtering out those that do not satisfy stated syntactic restrictions. These restrictions are given as Boolean expressions associated with the context-free rules; a tree is properly annotated only if all the Boolean expressions corresponding to the rules used in the analysis are simultaneously true. Again, using the rules of Figure 1,

```
(S (NP john)
   (VP (TENSE &past)
       (V go)))
```

is transformed into

[1]Of course, more sophisticated approaches to morphological analysis would seek to analyze the LEX relation more fully. See, for example, [56] and [54].

```
(S (NP: Proper
      john)
   (VP: ¬Transitive
      (TENSE &past)
      (V: ¬Transitive
         go)))
```

3.2.3 The Relation TRANSLATE

Logical-form synthesis rules are specified as augments to the context-free grammar. There is a language whose expressions denote translations (syntactic formulas); an expression from this language is attached to each context-free rule and serves to define the composite translation at a node in terms of the translations of its immediate constituents. In the sample sentence, TENSE' and V' (the translations of TENSE and V respectively) would denote the λ-expressions specified in their respective translation rules. VP' (the translation of the VP) is defined to be the value of (SAP (SAP COMP' TENSE') V'), where COMP' is a constant λ-expression and SAP is the *symbolic-application operator*. This works out to be (λ X (past (go X))). Finally, the symbolic application of VP' to NP' yields (past (go John)). (For convenience we shall henceforth use square brackets for SAP and designate (SAP α β) by $\alpha[\beta]$.)

Before describing the symbolic-application operator in more detail, it is necessary to explain the exact nature of the data objects serving as translations. At one level, it is convenient to think of the translations as λ-expressions, since λ-expressions are a convenient notation for specifying how fragments of a translation are substituted into their appropriate operator-operand positions in the formula being assembled--especially when the composition rules follow the syntactic structure as encoded in the parse tree. There are several phenomena, however, that require the storage of more information at a node than can be represented in a bare λ-expression. Two of the most conspicuous phenonema of this type are quantifier scoping and unbounded dependencies ("gaps").

Our approach to quantifier scoping has been to take a version of Cooper's storage technique, originally proposed in the context of model-theoretic semantics, [19] and adapt it to the needs of translation. For the time being, let us take translations to be ordered pairs whose first component (the *head*) is an expression in the target language, characteristically a λ-expression. The second component of the pair is an object called *storage*, a structured collection of sentential operators that can be applied to a sentence matrix in such a way as to introduce a quantifier and "capture" a free variable occurring in that sentence matrix.[2]

For example, the translation of "a happy man" might be < m , (λ S (some m (and (man m)(happy m)) S)) >.[3] Here the head is *m* (simply a free variable), and storage consists of the λ-expression (λ S ...). If the verb phrase "sleeps" were to receive the translation < (λ X (sleep X)), ϕ > (i.e., a unary predicate as head and no storage), then the symbolic application of the verb phrase translation to the noun phrase translation would compose the heads in the usual way and take the "union" of the storage yielding < (sleep m), (λ S (some m (and (man m)(happy m)) S)) >.

We define an operation called "pull.s," which has the effect of "pulling" the sentence operator out of storage and applying it to the head. There is another pull operation, pull.v, which operates on heads representing unary predicates rather than sentence matrices. When pull.s is applied in our example, it yields < (some m (and (man m)(happy m)) (sleep m)), ϕ >, corresponding to the translation of the clause "a happy man sleeps." Note that, in the process, the free variable m has been "captured." In model-theoretic semantics this capture would ordinarily be meaningless, although one can complicate the mathematical machinery to achieve the same effect. Since ***translation*** is fundamentally a ***syntactic*** process, however, this operation is well-defined and quite natural.

To handle gaps, we enriched the translations with a third component: a variable corresponding to the gapped position. For example, the translation of the relative clause "... [that] the man saw" would be a triple: < (past (see X Y)), Y, (λ S (the X (man X) S))>, where the second component, Y, tracks the free variable corresponding to the gap. At the node at which the gap was to be discharged, λ-abstraction would occur (as specified in the grammar by the operation "ungap"), thereby producing the unary predicate (λ Y (past (see X Y))), which

[2]In the sample grammar presented in Section 3.5.1, the storage-forming operation is notated *mk.mbd*.

[3]Following [91], a quantified expression is of the form (*quantifier, variable, restriction, body*)

would ultimately be applied to the variable corresponding to the head of the noun phrase.

It turns out that triples consisting of <head, var, storage> are adequate to serve as translations of a large class of phrases, but that the application operator needs to distinguish two subcases (which we call type A and type B objects). Until now we have been discussing type A objects, whose application rule is given (roughly) as

$$\langle \mathrm{hd,var,sto}\rangle[\langle \mathrm{hd',var',sto'}\rangle] = \langle(\mathrm{hd\ hd'}),\mathrm{var} \cup \mathrm{var'},\ \mathrm{sto} \cup \mathrm{sto'}\rangle \quad ,$$

where one of var or var' must be null. In the case of type B objects, which are assigned primarily as translations of determiners, the rule is

$$\langle \mathrm{hd,var,sto}\rangle[\langle \mathrm{hd',var',sto'}\rangle] = \langle \mathrm{var},\ \mathrm{var'},\ (\mathrm{hd\ hd'}) \cup \mathrm{sto} \cup \mathrm{sto'}\rangle \quad .$$

For example, if the meaning of "every" is

$$\mathrm{every'} = \langle(\lambda\ P\ (\lambda\ S\ (\mathrm{every}\ X\ (P\ X)\ S))),\ X,\ \phi\rangle$$

and the meaning of "man" is

$$\mathrm{man'} = \langle\ \mathrm{man},\ \phi,\ \phi\ \rangle \quad ,$$

then the meaning of "every man" is

$$\mathrm{every'[man']} = \langle\ X\ ,\ \phi,\ (\lambda\ S\ (\mathrm{man}\ X)\ S)\rangle \quad ,$$

as expected.

Nondeterminism arises in two ways. First, since pull operations can be invoked nondeterministically at various nodes in the parse tree (as specified by the grammar), there exists the possibility of computing multiple scopings for a single context-free parse tree. (See Section 3.3.2 for an example of this phenomenon.) In addition, the grammar writer can specify explicit nondeterminism by associating several distinct translation rules with a single context-free production. In this case, he can control the application of a translation schema by specifying for each schema a *guard*, i.e., a Boolean combination of features that the nodes analyzed by the production must satisfy for the translation schema to be applicable.

3.2.4 Implementation of a Translation System

The techniques presented in Sections 3.2.2 and 3.2.3 were implemented in a parsing and translation system called PATR-I, which was used as a component in a dialogue system discussed in Section 3.3.3. The input to the system is a sentence, which is preprocessed by a lexical analyzer. Parsing is performed by a simple recursive descent parser, augmented to add annotations to the nodes of the parse tree. Translation is then done in a separate pass over the annotated parse tree. Thus, the four conceptual phases are implemented as three actual processing phases. This folding of two phases into one was done purely for reasons of efficiency and has no effect on the actual results obtained by the system. Functions to perform the storage manipulation, gap handling, and the other features of translation presented earlier have all been realized in the translation component of the running system. The next section describes an actual grammar that has been used in conjunction with this translation system.

3.3. Experiments in Producing and Using Logical Form

3.3.1 A Working Grammar

To illustrate the ease with which diverse semantic features could be handled, a grammar was written that defines a semantically interesting fragment of English along with its translation into logical form [91]. The grammar for the fragment illustrated in this dialogue is compact, occupying only a few pages, yet it gives both syntax and semantics for modals, tense, aspect, passives, and lexically controlled infinitival complements. (A portion of the grammar is included as Section 3.5.1).[4] The full test grammar, loosely based on DIAGRAM [110] but restricted and modified to reflect changes in approach, was the grammar used to specify the translations of the sentences in the sample dialogue of Section 3.5.3.

[4]Since this is just a small portion of the actual grammar selected for expository purposes, many of the phrasal categories and annotations will seem unmotivated and needlessly complex. These categories and annotations *are* utilized elsewhere in the test grammar.

3.3.2 An Example of the System's Operation

The grammar presented in Section 3.5.1 encodes a relation between sentences and logical-form expressions. We now present a sample of this relation, as well as its derivation, with a sample sentence: "Every man persuaded a woman to go."

Lexical analysis relates the sample sentence to two morpheme streams:

- every man &ppl persuade a woman to go
- every man &past persuade a woman to go.

The first is immediately eliminated because there is no context-free parse for it in the grammar. The second, however, is parsed as

```
[S (SDEC (NP (DETP (DDET (DET every)))
             (NOM (NOMHD (NOUN (N man)))))
         (PREDICATE (AUXP (TENSE &past))
                    (VPP (VP (VPT (V persuade)))
                             (NP (DETP (A a))
                                 (NOM (NOMHD (NOUN (N woman)))))
                             (INFINITIVE (TO to)
                                         (VPP (VP (VPT (V go]
```

While parsing is being done, annotations are added to each node of the parse tree. For instance, the NP $\rightarrow$ DETP NOM rule includes the annotation rule AGREE(NP, DETP, Definite). AGREE is one of a set of macros defined for the convenience of the grammar writer. This particular macro invocation is equivalent to the Boolean expression Definite(NP) $\Leftrightarrow$ Definite(DETP). Since the DETP node itself has the annotation Definite as a result of the preceding annotation process, the NP node now acquires the annotation Definite as well. At the bottom level, the Definite annotation was derived from the lexical entry for the word "every."[5] The whole parse tree receives the following annotation:

```
[S (SDEC (NP: Definite
             (DETP: Definite
                 (DDET: Definite
                     (DET: Definite
                         every)))
```

[5]Note that, although the annotation phase was described and is implemented procedurally, the process actually used guarantees that the resulting annotation is exactly the one specified declaratively by the annotation rules.

```
        (NOM (NOMHD (NOUN (N man)))))
   (PREDICATE (AUXP (TENSE &past))
              (VPP (VP: Active
                        (VPT: Active,Transitive,TakesInf
                              (V: Active,Transitive,TakesInf
                                  persuade)))
                    (NP (DETP (A a))
                        (NOM (NOMHD (NOUN (N woman)))))
                    (INFINITIVE (TO to)
                        (VPP (VP: Active
                                  (VPT: Active
                                        (V: Active
                                            go]          .
```

Finally, the entire annotated parse tree is traversed to assign translations to the nodes through a direct implementation of the process described in Section 3.2.3. (Type A and B objects in the following examples are marked with a prefix 'A:' or 'B:'.) For instance, the VP node covering (persuade a woman to go), has the translation rule VPT'[NP'][INFINITIVE']. When this is applied to the translations of the node's constituents, we have

```
<A: (λ X (λ P (λ Y (persuade Y X (P X)))>
   [<A: X2, ϕ, (λ S (some X2 (woman X2) S))>]
      [<A: (λ X (go X))>]
```

which, after the appropriate applications are performed, yields

```
<A: (λ P (λ Y (persuade Y X2 (P X2)))), ϕ,
    (λ S (some X2 (woman X2) S))>
      [<A: (λ X (go X))>]

= <A: (λ Y (persuade Y X2 (go X2))), ϕ, (λ S (some X2 (woman X2) S))>     .
```

After the past operator has been applied, we have

```
<A: (λ Y (past (persuade Y X2 (go X2)))), ϕ, (λ S (some X2 (woman X2) S))>
.
```

At this point, the pull operator (pull.v) can be used to bring the quantifier out of storage, yielding[6]

```
<A: (λ Y (some X2 (woman X2) (past (persuade Y X2 (go X2))))), ϕ, ϕ>     .
```

This will ultimately result in "a woman" getting narrow scope. The other alternative is for the

[6]For convenience, when a final constituent of a translation is ϕ it is often not written. Thus we could have written <A: (λ Y (some ...) ...)> in this case.

quantifier to remain in storage, to be pulled only at the full sentence level, thus resulting in the other scoping. In Figure 2 we have added the translations to all the nodes of the parse tree. Nodes with the same translations as their parents were left unmarked. After examination of the S node translations, the original sentence is given the fully scoped translations

```
(every X2 (man X2)
          (some X1 (woman X1) (past (persuade X2 X1 (go X1)))))
```

and

```
(some X1 (woman X1)
         (every X2 (man X2) (past (persuade X2 X1 (go X1)))))
```

.

3.3.3 A Simple Question-Answering System

As mentioned in Section 3.1, we were able to demonstrate the semantic capabilities of our language system by assembling a small question-answering system. Our strategy was to first translate English into logical formulas of the type discussed in [91], which were then postprocessed into a form suitable for a first-order deduction system. We used a connection graph theorem prover, described in Section 8 of this report. (Another possible approach would have been to translate directly into first-order logic, or to develop direct proof procedures for the non-first-order language.) Thus, we were able to integrate all the components into a question-answering system by providing a simple control structure that accepted an input, translated it into logical form, reduced the translation to first-order logic, and then either asserted the translation in the case of declarative sentences or attempted to prove it in the case of interrogatives. (Only yes/no questions have been implemented.)

The main point of interest is that our question-answering system was able to handle complex semantic entailments involving tense, modality, etc.—and that, moreover, it was not restricted to extensional evaluation in a database, as with conventional question-answering systems. For example, our system was able to handle the entailments of sentences like

- John could not have been persuaded to go.

(The transcript of a sample dialogue is included as Section 3.5.3.)

```
(S: <A: (past (persuade X1 X2 (go X2))), φ,
        (λ S (every X1 (man X1) S))
        (λ S (some X2 (woman X2) S))>,
    <A: (some X2 (woman X2) (past (persuade X1 X2 (go X2)))), φ,
        (λ S (every X1 (man X1) S))>
    <A: (every X2 (man X2)
               (some X1 (woman X1) (past (persuade X2 X1 (go X2)))))>
    <A: (some X1 (woman X1)
              (every X2 (man X2) (past (persuade X2 X1 (go X2)))))>
  (SDEC
    (NP: <A: X1, φ, (λ S (every X1 (man X1) S))>
      (DETP: <B: (λ P (λ S (every X (P X) S))),
               X>
        (DDET (DET every)))
      (NOM: <A: (λ X (man X))>
        (NOMHD (NOUN (N man)))))
    (PREDICATE: <A: (λ X (past (persuade Y X2 (go X2)))), φ,
                    (λ S (some X2 (woman X2) S))>,
                <A: (λ X (some X2 (woman X2)
                                  (past (persuade Y X2 (go X2))))),
                   φ, φ>
      (AUXP: <A: (λ P (λ X (past (P X))))>
        (TENSE &past))
      (VPP: <A: (λ Y (persuade Y X2 (go X2))), φ,
                (λ S (some X2 (woman X2) S))>
        (VP (VPT: <A: (λ X
                         (λ P
                            (λ Y (persuade Y X (P Y))))>
              (V persuade)))
          (NP: <A: X2, φ, (λ S (some X2 (woman X2) S))>
            (DETP: <B: (λ P (λ S (some X (P X) S))),
                     X>
              (A a))
            (NOM: <A: (λ X (woman X))>
              (NOMHD (NOUN (N woman)))))
          (INFINITIVE (TO: none
                           to)
                      (VPP: <A: (λ X (go X))>
                         (VP (VPT (V go]
```

Figure 2: Node-by-node translation of a sample sentence

The reduction of logical form to first-order logic (FOL) was parameterized by a set of recursive expansions for the syntactic elements of logical form in a manner similar to Moore's use of an axiomatization of a modal language of belief [90]. For example, (past P) is expanded,

```
INPUT: every man must be happy
LF:    (every X (man X)
              (necessary (and (happy X)
                              (thing X))))
FOL:   (every x0172
             (implies (man REALWORLD x0172)
                 (every w0173
                       (implies (poss REALWORLD w0173)
                               (and (happy w0173 x0172)
                                   (thing w0173 x0172))))))

INPUT: bill persuaded john to go
LF:    (past (persuade bill john (go john)))
FOL:   (some w0175
            (and (past w0175 REALWORLD)
                (some w0176
                     (and (persuade w0175 bill john w0176)
                         (go w0176 john)))))
```

Figure 3: Translation to LF and Reduction to FOL

with respect to a possible world w, as

(some w2 (and (past w2 w) <P,w2>)) ,

where "<P,w2>" denotes the recursive FOL reduction of P relative to the world w2. The logical form that was derived for the sample sentence "John went" therefore reduces to the first-order sentence

(some w (and (past w REALWORLD) (go w John))) .

More complicated illustrations of the results of translation and reduction are shown in Figure 3. Note, for example, the use of restricted quantification in LF and ordinary quantification in FOL.

To compute the correct semantic entailments, the deduction system was preloaded with a set of meaning postulates (axioms) giving inferential substance to the predicates associated with lexical items (see Section 3.5.2).

3.4. Further Extensions

We are continuing to refine the grammar formalism and improve the implementation. Some of the refinements are intended to make the annotations and translations easier to write. Examples include

- Allowing nonbinary features, along with sets of values, in the annotations and guards (extending the language to include equality and set operations).
- Generalizing the language used to specify synthesis of logical forms and developing a more uniform treatment of translation types.
- Generalizing the "gap" variable feature to handle arbitrary collections of designated variables by using an "environment" mechanism. This is useful in achieving a uniform treatment of free word order in verb complements and modifiers.

In addition, we are working on extensions of the syntactic machinery, including phrase-linking grammars to handle displacement phenomena [100], and methods for generating the augmented phrase-structure grammar through a metarule formalism similar to that of [61]. We have also experimented with alternative parsing algorithms, including a chart parser [6] adapted to carry out annotation and translation in the manner described in this paper.

3.5. Example

3.5.1 Sample Grammar Rules

The following is a portion of a test grammar for the PATR-I English translation system. Only those portions of the grammar utilized in analyzing the sample sentences in the text were included. The full grammar handles the following constructs: modals, adjectivals, tense, predicative and nonpredicative copulatives, adverbials, quantified noun phrases, aspect, NP, PP, and infinitival complements, relative clauses, yes/no questions, restricted wh-questions, noun-noun compounds, passives, and prepositional phrases as predicates and adjectivals.

```
==================== Grammar Rules ====================

Constant EQ' = curry (LAMBDA (X Y) (equal X Y))

Constant PASS' =
<A: (LAMBDA P (LAMBDA X ((P X) Y))), NIL,
    (MK.MBD (QUOTE (LAMBDA S (some Y (thing Y) S)))) >

Constant PASSINF' =
<A: (LAMBDA P (LAMBDA I (LAMBDA X (((P X) I) Y)))), NIL,
    (MK.MBD (QUOTE (LAMBDA S (some Y (thing Y) S)))) >

AUXP -> TENSE;
  Translation:
    TENSE'

DDET -> DET;
  Annotation:
    [ Definite(DDET) ]
  Translation:
    DET'

DETP -> A;
  Annotation:
    [ ¬Definite(DETP) ]
  Translation:
    A'

DETP -> DDET;
  Annotation:
    [ AGREE(DETP, DDET, Definite) ]
  Translation:
    DDET'

INFINITIVE -> TO VPP;
  Annotation:
    [ AGREE(INFINITIVE, VPP, Gappy, Wh) ]
  Translation:
    pull.v(VPP')

NOM -> NOMHD;
  Annotation:
    [ AGREE(NOM, NOMHD, Gappy) ]
  Translation:
    NOMHD'

NOMHD -> NOUN;
```

```
  Translation:
    NOUN'

NOUN -> N;
  Translation:
    N'

NP -> DETP NOM;
  Annotation:
    [ AGREE(NP, NOM, Gappy) ]
    [ Predicative(NP) ∨ ¬Predicative(NP) ]
    [ AGREE(NP, DETP, Definite) ]
  Translation:
    ¬Predicative(NP): DETP'[NOM']
    Definite(NP) & Predicative(NP): EQ'[DETP'[NOM']]
    ¬Definite(NP) & Predicative(NP): NOM'

PREDICATE -> AUXP VPP;
  Annotation:
    [ AGREE(PREDICATE, VPP, Active, Gappy, Wh) ]
  Translation:
    pull.v(AUXP'[VPP'])

S -> SDEC;
  Annotation:
    [ ¬Gappy(SDEC) ]
    [ ¬Wh(SDEC) ]
  Translation:
    SDEC'

SDEC -> NP PREDICATE;
  Annotation:
    [ Gappy(NP) ∨ Gappy(PREDICATE) ⇔ Gappy(SDEC) ]
    [ ¬Predicative(NP) ]
    [ Wh(NP) ∨ Wh(PREDICATE) ⇔ Wh(SDEC) ]
    [ ¬(Gappy(NP) & Gappy(PREDICATE)) ]
  Translation:
    pull.s(PREDICATE'[NP'])

VP -> VPT;
  Annotation:
    [ ¬Transitive(VPT) ]
    [ ¬TakesInf(VPT) ]
    [ Active(VPT) ]
    [ Active(VP) ]
  Translation:
    VPT'
```

```
VP -> VPT NP INFINITIVE;
  Annotation:
    [ TakesInf(VPT) ]
    [ Transitive(VPT) ]
    [ ¬Predicative(NP) ]
    [ AGREE(VP, VPT, Active) ]
    [ Wh(NP) ∨ Wh(INFINITIVE) ⇔ Wh(VP) ]
    [ IF(Active(VPT),
         ((Gappy(NP) ∨ Gappy(INFINITIVE)) ⇔ Gappy(VP)),
           & ¬(Gappy(NP) & Gappy(INFINITIVE)),
         (¬Gappy(VPT) & Gappy(NP))) ]
  Translation:
    Active(VP):  pull.v(VPT'[NP'][INFINITIVE'])
    ¬Active(VP):  pull.v(PASSINF'[VPT'][INFINITIVE'])

VPP -> VP;
  Annotation:
    [ AGREE(VPP, VP, Gappy, Wh) ]
    [ Active(VP) ]
  Translation:
    VP'

VPT -> V;
  Annotation:
    [ AGREE(VPT, V, Active, Transitive, TakesInf) ]
  Translation:
    V'


======================= Lexicon =======================

N -> man;
  Translation:
    <A: man, NIL, NIL >

N -> woman;
  Translation:
    <A: woman, NIL, NIL >

DET -> every;
  Annotation:
    [ Definite(DET) ]
  Translation:
    <B: (LAMBDA P (LAMBDA S (every X (P X) S))), X, NIL >

A -> a;
```

```
  Translation:
    <B: (LAMBDA P (LAMBDA S (some X (P X) S))), X, NIL >

V -> persuade;
  Annotation:
    [ Transitive(V) ]
    [ Active(V) ∨ ¬Active(V) ]
    [ TakesInf(V) ]
  Translation:
    curry (LAMBDA (X P Y) (persuade Y X (P X)))

V -> go;
  Annotation:
    [ ¬Transitive(V) ]
    [ ¬TakesInf(V) ]
    [ Active(V) ]
  Translation:
    <A: go, NIL, NIL >

TENSE -> &past;
  Translation:
    curry (LAMBDA (P X) (past (P X)))
```

3.5.2 Meaning Postulates

```
[every w (every u (iff (past w u)
                       (not (past u w]
(every w (some u (past w u)))
[every w (every x (every y (every z (implies (promise w x y z)
                                             (past w z]
[every w (every x (every y (every z (implies (persuade w x y z)
                                             (past w z]
(every w (every x (thing w x)))
[every w (every x (every z (implies (want w x z)
                                    (past w z]
(every w (poss w w))
[every w (every u (implies (past w u)
                           (poss w u]
[every w (every u (every v (implies (and (past1 w u)
                                         (past1 u v))
                                    (past2 w v]
[every w (every z (implies (past2 w z)
                           (past w z]
[every w (every z (iff (past w z)
(past1 w z]
```

3.5.3 Transcript of Sample Dialogue

```
>> john is happy
OK.

>> is john happy
Yes.

>> is john a happy man
I don't know.

>> john is a man
OK.

>> is john a happy man
Yes.

>> no man could have hidden a book
OK.

>> did john hide a book
No.

>> bill hid a book
OK.

>> is bill a man
No.

>> was john a man
I don't know.

>> every man will be a man
OK.

>> will john be a man
Yes.

>> bill persuaded john to go
OK.

>> could john have been persuaded to go
```

```
Yes.

>> will john be persuaded to go
I don't know.
```

4. The Formalism and Implementation of PATR-II

This section was written by Stuart Shieber, Hans Uszkoreit, Fernando Pereira, Jane Robinson, and Mabry Tyson.

4.1. The PATR-II Formalism

4.1.1 Motivation for the Formalism

The goal of natural-language processing is simple: to enable computers to participate in dialogues with humans in their language in order to make the computers more useful to their creators. The pursuit of this objective, however, has been a difficult task for at least two reasons: first, the phenomenon of human language is not as well understood as is popularly supposed; second, the tools for teaching computers what we do know about human language are still quite primitive. The solution of these problems falls into the research domains of linguistics and computer science, respectively.

Similar problems have previously arisen in the field of computer science. With the advent of digital computers, the need for effective ways of communicating with computers, other than by means of patch panels, became quickly evident. The "black art" of programming-language design has improved greatly over the years and much is now known about effective communication with computers. In particular, the criteria for good programming languages are their *power*, *utility*, and, in the case of research languages, *simplicity and mathematical well-foundedness*. Note that only the first of these can be measured objectively; in fact, the power of most current programming languages is equivalent to that of a Turing machine. However, the basic fact is that more power is considered better. On the other hand, the other two criteria are inherently subjective, which is why programming language design is an art rather than a science. Utility, in fact, is usually a relative measure, relative to the purposes the language is designed for. SNOBOL is a useful language for string manipulation, but awkward at best for, say, matrix manipulation. This is because the primitives supplied by SNOBOL do not match the common underlying operations of matrix handling.

Among the evaluation criteria for programming languages that have been assiduously promoted in the recent past is the aforementioned criterion of simplicity. Other trends have been in the direction of declarative languages, languages emphasizing structured programming and modularity, and the like. The design of a grammar formalism embodies the same problems as the design of a programming language simply because it aspires to the same goal, i.e., effective communication of information to a computer. Thus, the same criteria can be applied: the formalism should be as powerful as possible, should incorporate the types of primitives that natural-language grammar writers find they need, should be simple and mathematically well-founded. Trends from programming-language design, such as declarativeness and modularity, can also be applied to the problem of designing grammar formalisms for computers.

Theoretical linguists have been concerned with designing grammatical formalisms that provide the tools for expressing universal and language-specific generalizations in a concise and transparent fashion. One of their main objectives in this task is to constrain the power of their formalisms in concurrence with the cross-linguistic set of constraints upon syntactic and semantic phenomena that are found in natural language.

A radical but widespread opinion regarding the choice of an appropriate formalism is that it should embody all nonaccidental regularities that are observed in all languages, i.e., those that belong to universal grammar. For instance, if all languages are thought to possess coordination, this fact should derive from the formalism. If, on the other hand, no language in the world has the word "famakupa," which would be phonologically well-formed in many languages, we can then regard this observation as an accidental fact that will be represented in the set of particular grammars.

The PATR-II formalism as a tool for grammar writing does not attempt to encode most of the statements of universal grammar. It is based on the generally accepted view that sentences have structure, and it provides for structures that are more complex than phrase structure trees. Not only do the regularities of specific languages have to be encoded by the user of the formalism—either in the proposed rules or in stipulations with regard to usage constraints—but so do most cross-linguistic generalizations, including constraints on generative power. The cross-linguistic generalization and constraints can be reflected in a selected implementation or usage notation. We shall discuss an example of such a notation later.

4.1.2 Design of the Formalism

We now describe the formalism that underlies the implementations of PATR-II. In some sense, this is the "operational semantics" of a PATR-II grammar. Certain implementations may make use of certain abbreviations or conventions, but the operation of such implementations is defined in terms of this simple underlying formalism. Thus, the formalism bears the same relation to PATR-II implementations as, say, pure LISP does to MACLISP.

The basic operation in PATR-II is *unification*, an extended pattern-matching technique that was first used in logic and theorem-proving research and has been arousing considerable interest of late in the linguistics community. Rather than unifying logic terms, however, PATR unification operates on directed acyclic graphs (DAG).[1] DAGs can be atomic symbols or sets of label/value pairs whose labels (also called attributes or features) are atomic symbols or other DAGs (i.e., subDAGs). Since two labels can point to the same DAG, the term graph is used rather than tree. DAGs are notated either by drawing the labeled graph structure itself or, as in this paper, notating the sets of label/value pairs in square brackets ([ ]), with the labels separated from their values by a colon (:), e.g.,

```
[cat: v
 head: [aux: false
        form: nonfinite
        voice: active
        trans: [pred: eat
               arg1: <f1134>
                     []
               arg2: <f1138>
```

[1]Technically, these are rooted, unordered, directed, acyclic graphs with labeled arcs. See Appendix A for a more formal definition of PATR-II grammars and accompanying notions.

```
                              []]]
      syncat: [first: [cat: np
                       head: [trans: <f1134>]]
               rest: [first: [cat: np
                              head: [trans: <f1138>]]
                      rest: <f1140>
                            lambda]
               tail: <f1140>]]
```

Note that the re-entrant structure, where two arcs point to the same node, is notated by labeling the DAG with an arbitrary label (in angle brackets (< >)) and then using that label for future references to the DAG.

Associated with each lexical entry in the lexicon is a set of DAGs.[2] The root of each DAG will have an arc labeled *cat* whose value will be the category of the associated lexical entry. Other arcs may encode information about the syntactic features, translation, or syntactic subcategorization of the entry.

PATR-II grammars consist of rules with a context-free phrase structure portion and a series of unifications on the DAGs associated with the constituents taking part in the use of the rule. The grammar rules notate how constituents can be built up to form new constituents with associated DAGs. The right side of the rule lists the *cat* values of the DAGs associated with the child constituents; the left side, the *cat* of the parent. Other unifications specify equivalences that must exist among the various DAGs and subDAGs of the parent and children. Thus, the formalism uses only one representation (DAGs) for lexical, syntactic, and semantic information, and just one operation (unification) on this representation.

By way of example, we present a small grammar for a fragment of English, accompanied by a lexicon associating words with DAGs.

$S \rightarrow NP\ VP$

$$<VP\ agr> = <NP\ agr>$$

$VP \rightarrow V\ NP$

$$<VP\ agr> = <V\ agr>$$

[2]We shall postpone until later any discussion as to how this association is encoded or implemented.

Uther:

$$\begin{aligned} \langle cat\rangle &= np \\ \langle agr\ number\rangle &= singular \\ \langle agr\ person\rangle &= third \end{aligned}$$

Arthur:

$$\begin{aligned} \langle cat\rangle &= np \\ \langle agr\ number\rangle &= singular \\ \langle agr\ person\rangle &= third \end{aligned}$$

knights:

$$\begin{aligned} \langle cat\rangle &= v \\ \langle agr\ number\rangle &= singular \\ \langle agr\ person\rangle &= third \end{aligned}$$

This grammar (plus lexicon) admits the two sentences "Uther knights Arthur" and "Arthur knights Uther." The phrase structure associated with the first of these is

$[_S\ [_{NP}\ \text{Uther}]\ [_{VP}\ [_V\ \text{knights}]\ [_{NP}\ \text{Arthur}]]]$

The VP rule requires that the *agr* feature of the DAG associated with the VP be the same as (unified with) the *agr* of the V. Thus the VP's *agr* feature will have as its value the *same* node as the V's *agr* and, hence, the same values for the *person* and *number* features. Similarly, by the unification associated with the S rule, the NP will have the same *agr* value as the VP and, consequently, the V. We have thus encoded a form of subject-verb agreement.

4.1.3 Power of the Formalism

PATR-II grammars, as just presented, are extremely powerful; in fact, they are equivalent to Turing machines. We therefore present a straightforward constraint upon their power that guarantees decidability, a constraint Pereira [98] calls *off-line parsability*. Off-line parsability requires that there be no nonproductive recursive chains of rules in the grammar, i.e., chains that can consume no input. Recursive chains of unary rules, or chains of rules in which all but one nonterminal in each rule can derive the empty string, are thus eliminated. In the case of context-free grammars, removing such rules does not change the power of the formalism.

PATR-II grammars, however, are restricted by this constraint—the specific effect of which is to render the parsing problem decidable.

Nonetheless, the power of PATR-II grammars remains great. Appendix B presents grammars for the non-context-free triple-counting language $a^n b^n c^n$ and the non-indexed language $a^{2^{2^n}}$. It remains an open question whether there are interesting further constraints on PATR-II and other unification-based formalisms that reduce the parsing problem significantly without unduly constricting generative capacity. We should keep in mind that evaluation of such constraints requires aesthetic judgments, not scientific ones.

4.1.4 Future Research to Improve the Formalism

The formalism is broad and powerful enough to handle most—indeed, probably all—phenomena in the syntax and semantics of natural language. It has also turned out to be well suited for the classes of phenomena considered so far. Most of the research will have to be done in the area of choosing appropriate strategies for application of the formalism. However, there is a class of phenomena that might justify some extension or modification of the formalism: the phenomena of free or variable word order.

Although the formalism is powerful enough to deal with word order variability, there is a strong feeling on our side that it should be possible to express free variation more directly. We plan to work out the necessary modifications in the near future and, to this end, we hope to be able to use results of a proposed parallel research project for studying such word order variation.

One direction in which the formalism might be extended to allow for word order variability is the relaxation of constraints on possible feature values. Let us assume that these values can be nonatomic, i.e., that they can be sets or sequences. Let us furthermore regard the permutations of verb complements as an example of order variation. By allowing structure in the feature system, we can encode much more information about possible VP structures. One example follows, but the possibilities are endless (literally so, since, by doing this, we move from the realm of context-free languages to the realm of Turing computability). Suppose the range R of the Syncat [see Section 4.3.4.1.] feature included atomic symbols and all sets and sequences of elements of R. Also suppose that we define an operation $\ominus$ acting on compound elements of R such that

$$< \alpha, \beta_1, \ldots, \beta_n > \ominus \gamma = < \alpha \ominus \gamma, \beta_1, \ldots, \beta_n >$$
$$\{\alpha, \beta_1, \ldots, \beta_n\} \ominus \gamma = < \alpha \ominus \gamma, \{\beta_1, \ldots, \beta_n\} >$$

Now we can write a grammar as follows:

$$VP \rightarrow VP\,COMP$$
$$Syncat(VP_1) = Syncat(VP_2) \ominus Form(COMP)$$
$$VP \rightarrow V$$
$$Syncat(VP) = Syncat(V)$$

This structuring allows us a way of expressing free word order in the subcategorizations. Thus, if a verb subcategorizes for $\{< \alpha\beta > \gamma\}$, it allows argument structures of $\alpha\beta\gamma$ and $\gamma\alpha\beta$ but not $\beta\alpha\gamma$ or $\alpha\gamma\beta$. Using similar techniques, ID/LP could be encoded in a $\ominus$ operator working on complex structures. In fact, this is basically how the ID/LP direct parsing algorithm of Shieber [117] works.

4.2. Some Uses of the Formalism: The Current PATR-II Grammar Design

In this section we present some ideas concerning different uses of the formalism and describe our own current usage. Although most of the techniques presented here represent our current use of the basic formalism, they should not be identified with the formalism itself, which allows for quite different strategies of grammar writing.

It should be mentioned that many syntactic constructs not discussed in this introduction to the formalism are currently handled by existing grammars for our implemented system. Among these are $\overline{S}$ complements, active, passive, "there" insertion, extraposition, raising and equi constructions, etc. (See Appendix D for a more complete PATR-II grammar, Appendix E for a transcript of the parsing system using the grammar.)

Before we start explaining our use of the formalism let us emphasize once more the considerable freedom it allows for writing a grammar. The only label with any special significance in the formalism is the arc label *cat*. This is a consequence of the decision to use traditional context-free phrase structure rules to create part of the syntactic and semantic structure. Everything else, including the appropriate category symbols, has to be designed by the grammar writer. Part of the process of writing a grammar, therefore, involves deciding on a set of arc labels (attributes, features) that are used to encode pertinent information about constituents.

4.2.1 Feature Percolation

Linguistic formalisms often provide a technique for percolating a large set of features from a given child to its parent, for instance, by means of the *head feature convention* in GPSG or the $\uparrow=\downarrow$ equation in LFG. Grouping of features in this way can be accomplished in PATR-II by placing the features on a subDAG of the DAG of the child under a special attribute, say *head*, and then unifying the *head* attribute of parent and child with a unification of the form *<parent head>* = *<child head>*. Agreement features and case, are examples of features that could be percolated in this way. Thus, the previous sample grammar might be extended to allow *head* feature percolation as follows:

```
S → NP VP
          <S head>  =  <VP head>
      <NP head agr>  =  <S head agr>

VP → V NP
          <VP head>  =  <V head>

Uther:
                <cat>  =  np
  <head agr number>  =  singular
   <head agr person>  =  third

Arthur:
                <cat>  =  np
  <head agr number>  =  singular
   <head agr person>  =  third

knights:
                <cat>  =  v
```

$$\begin{aligned} \langle \textit{head agr number} \rangle &= \textit{singular} \\ \langle \textit{head agr person} \rangle &= \textit{third} \end{aligned}$$

4.2.2 Semantics

The meaning of a constituent, a segment of logical form, needs to be recorded somewhere in the DAG associated with it. For reasons of modularity, we would like this encoding to be separable from the other portions of the DAG that encode syntactic information. To encode meanings with no extra apparatus, we shall use the following encoding of logical-form fragments. A predicate applied to several arguments, for instance—$f(a, b, c)$—will be encoded with the arcs *pred* and arg_i, respectively. A constant will be notated with the feature *ref*. Thus, the fragment above would be encoded as

```
[pred:  f
 arg1:  [ref: a]
 arg2:  [ref: b]
 arg3:  [ref: c]]
```

More evocative names for the argument positions could be used, e.g., *agent, patient, goal,* though we will not use them here.

Note that the translation of a parent constituent is often associated with the translation of a specific child constituent (with other child translations adding further information). For instance, the translation of a VP will be identical to that of the child V, with complements supplying translations assigned to the arguments. We can therefore make *trans* a *head* feature and allow the standard *head* feature mechanism to distribute it appropriately. Adding translations to our small grammar, we get:

$S \rightarrow NP\ VP$

$$\begin{aligned} \langle \textit{S head} \rangle &= \langle \textit{VP head} \rangle \\ \langle \textit{NP head agr} \rangle &= \langle \textit{S head agr} \rangle \\ \langle \textit{S head trans arg1} \rangle &= \langle \textit{NP head trans} \rangle \end{aligned}$$

$VP \rightarrow V\ NP$

$$\begin{aligned} \langle \textit{VP head} \rangle &= \langle \textit{V head} \rangle \\ \langle \textit{VP head trans arg2} \rangle &= \langle \textit{NP head trans} \rangle \end{aligned}$$

Uther:

$$\begin{aligned} \langle cat \rangle &= np \\ \langle head\ agr\ number \rangle &= singular \\ \langle head\ agr\ person \rangle &= third \\ \langle head\ trans\ ref \rangle &= uther' \end{aligned}$$

Arthur:

$$\begin{aligned} \langle cat \rangle &= np \\ \langle head\ agr\ number \rangle &= singular \\ \langle head\ agr\ person \rangle &= third \\ \langle head\ trans\ ref \rangle &= arthur' \end{aligned}$$

knights:

$$\begin{aligned} \langle cat \rangle &= v \\ \langle head\ agr\ number \rangle &= singular \\ \langle head\ agr\ person \rangle &= third \\ \langle head\ trans\ pred \rangle &= knight' \end{aligned}$$

This grammar will admit the same sentences as previously, yielding the translations (in prefix notation) *knight'(uther', arthur')*, and *knight'(arthur', uther')* respectively.

4.2.3 Coordinating Syntax and Semantics

The previous grammar performs a *de facto* coordination of syntax and semantics by requiring that the (syntactically) preverbal NP play the (semantic) role of first argument, and that the postverbal complement play the role of second argument. Such a direct one-time mapping is difficult to maintain, and various theories have solved this problem in different ways. In general, the solution requires adding one more *degree of freedom* in the mapping. GPSG obtains this degree of freedom because intensional-logic operators are able to act as combinators, reordering arguments. These operators are introduced through metarules (though they could have been introduced by lexical rules). LFG uses an intermediate representation to provide the additional degree of freedom, mapping syntactic objects onto a set of arbitrary labels—*SUBJ, OBJ, OBJ2*, etc.—and then mapping these in turn to argument positions.

Either of these solutions could be modeled in PATR-II, though our actual technique (which will be presented after subcategorization is discussed) is slightly different from both.

We offer an example of the LFG style solution at this juncture. An LFG grammar unifies the preverbal and postverbal NPs as the values of *subject* and *object*, respectively. If one declares these to be *head* features, they will be unified with the *subject* and *object* features of the V itself. The lexical entry for the V will then perform the second half of the mapping, i.e., from grammatical function to argument position.

S → *NP VP*

$$\begin{array}{rcl} \langle S\ head\rangle & = & \langle VP\ head\rangle \\ \langle NP\ head\ agr\rangle & = & \langle S\ head\ agr\rangle \\ \langle S\ head\ subject\rangle & = & \langle NP\ head\rangle \end{array}$$

VP → *V NP*

$$\begin{array}{rcl} \langle VP\ head\rangle & = & \langle V\ head\rangle \\ \langle VP\ head\ object\rangle & = & \langle NP\ head\rangle \end{array}$$

Uther:

$$\begin{array}{rcl} \langle cat\rangle & = & np \\ \langle head\ agr\ number\rangle & = & singular \\ \langle head\ agr\ person\rangle & = & third \\ \langle head\ trans\ ref\rangle & = & uther' \end{array}$$

Arthur:

$$\begin{array}{rcl} \langle cat\rangle & = & np \\ \langle head\ agr\ number\rangle & = & singular \\ \langle head\ agr\ person\rangle & = & third \\ \langle head\ trans\ ref\rangle & = & arthur' \end{array}$$

knights:

$$\begin{array}{rcl} \langle cat\rangle & = & v \\ \langle head\ agr\ number\rangle & = & singular \\ \langle head\ agr\ person\rangle & = & third \\ \langle head\ trans\ pred\rangle & = & knight' \\ \langle head\ trans\ arg1\rangle & = & \langle subject\ trans\rangle \\ \langle head\ trans\ arg2\rangle & = & \langle object\ trans\rangle \end{array}$$

It is now clear how a lexical rule might be written for passivization: it merely changes the roles of *subject* and *object* in the lexical entry in the appropriate way. The ability to perform such redirection of grammatical and semantic functions provides the requisite extra degree of freedom. Before presenting an alternative solution to the degree-of-freedom problem, we must discuss the related problems of verb phrase structure and subcategorization.

4.2.4 Verb Phrase Structure and Subcategorization

4.2.4.1 Nested versus Flat Structure

Various alternatives have been suggested for handling verb phrase structures in a grammar for English. The proposed methods fall into two main categories: flat structure and nested structure. The flat structure is epitomized by the treatment in GPSG. We shall start with this technique.

Suppose we have a GPSG of the form

$$< 1, VP \rightarrow V\ \alpha_1 \cdots \alpha_m >$$
$$< 2, VP \rightarrow V\ \beta_1 \cdots \beta_n >$$
etc.

This grammar generates flat verb-phrase structures in which the verb and all of its complements are siblings. In GPSG we get appropriate subcategorization by associating with the rule some distinguishing feature (in the nontechnical sense) then associating that feature with any verbs that subcategorize for the rule. (This association acts like a virtual pointer between verbs and rules.) The feature in the case of GPSG is the rule number.

Two points deserve mention. First, the rule number technique in GPSG is *outside* the feature system.[3] But, since there is presumably only a finite number of verb phrase rules, there is no reason that the rule number could not have status as a normal feature (in the technical sense). A PATR-II grammar using this technique would look like this:

$$VP \rightarrow V\,\alpha_1 \cdots \alpha_m$$
$$<V\ syncat> \ = \ 1$$
$$VP \rightarrow V\,\beta_1 \cdots \beta_n$$
$$<V\ syncat> \ = \ 2$$

and so on.

[3]Actually, the most recent versions of GPSG have abandoned the distinction between rule numbers and features.

Second, rule numbers are only one way of distinguishing rules. Any other distinguishing feature of rules could be used. In particular, if no two rules share the same right-hand side, the right-hand sides could themselves be used as the subcategorization, as in the following grammar:

$$\begin{aligned} VP &\rightarrow V\,\alpha_1 \cdots \alpha_m \\ &\quad <V\ syncat> = [\alpha_1, \ldots, \alpha_m] \\ VP &\rightarrow V\,\beta_1 \cdots \beta_n \\ &\quad <V\ syncat> = [\beta_1, \ldots, \beta_n] \end{aligned}$$

Of course, we have introduced notation here that is not found in the PATR-II formalism, namely, lists. Before explaining this, let us be even more free with notation. We could make the grammar still more concise by taking advantage of the fact that DAGs carry their category "on their sleeve," so to speak.

$$\begin{aligned} VP &\rightarrow V\,\alpha_1 \cdots \alpha_m \\ &\quad <V\ syncat> = \bigoplus_{i=1}^{m}[< \alpha_i\ cat >] \\ VP &\rightarrow V\,\beta_1 \cdots \beta_n \\ &\quad <V\ syncat> = \bigoplus_{i=1}^{n}[< \beta_i\ cat >] \end{aligned}$$

where $\bigoplus$ denotes the repeated use of the list concatenation operator $\oplus$.

Note that all the unifications of the rules in this sample VP grammar are of exactly the same form. We can take advantage of that fact in a grammar in which there is only one VP rule by making use of a regular expression notation for the right-hand side of the rule.

$$\begin{aligned} VP &\rightarrow V\,\{\alpha_1 \cup \alpha_2 \cup \cdots \cup \beta_m\}^* \\ &\quad <V\ syncat> = \bigoplus_{i=1}^{n}[< COMP_i\ cat >] \end{aligned}$$

where n is the number of constituents in the instantiation of the rule, and $COMP_i$ provides a way of accessing the constituents.

We can now begin to clarify just how such a free-wheeling subcategorization scheme can be implemented in strict PATR-II. First of all, the method of getting the behavior of a Kleene star in context-free grammars is to use a recursive category, i.e., for each possible complement category α_i, we have a rule:

$$VP_1 \rightarrow VP_2\ \alpha_i$$

$$<VP_2\ syncat> \ = \ <\alpha_i\ cat> \oplus <VP_1\ syncat>$$

We add a rule to start the recursion:

$$VP \rightarrow V$$

$$<VP\ syncat> \ = \ <V\ syncat>$$

We merely require that a "full-fledged" VP is one whose *syncat* is the empty list Λ. It can be easily proved that this grammar weakly generates the same language the previous one(s) did. The difference, of course, is that the structure is now nested, not flat.

Finally, the question remains as to how lists and the ⊕ operation can be encoded. Lists can be encoded recursively as either a special symbol denoting the empty list, Λ, or pairs containing a list element and a list. We shall call these two parts *first* and *rest*. The *syncat* arc of a verb will then have a value something like

```
[first: α1
 rest:  [first: α2
          rest: ···
                 [first: αm
                  rest:  Λ]···]]
```

The previous grammar can now be expressed as

$$VP_1 \rightarrow VP_2\ \alpha_i$$

$$<VP_2\ syncat\ first> \ = \ <\alpha_i\ cat>$$
$$<VP_1\ syncat> \ = \ <VP_2\ syncat\ rest>$$

$$VP \rightarrow V$$

$$<VP\ syncat> \ = \ <V\ syncat>$$

We have seen a smooth progression from flat to nested structure to deal with the same problem of subcategorization. The progression involved moving the information about constituency from phrase structu- rules to subcategorization information in the lexicon. Indeed, any context-free grammar can undergo such a transformation to yield an equivalent PATR-II grammar that has only one nonunary rule and preserves the weak-generative character of the language. (See Appendix C.) In effect, we just move all the syntactic information into the lexical entries, so that the same PATR-II grammar skeleton can be used to model any CF grammar. Because the construction is local, the two methods can be combined freely. It is this aspect that we take advantage of in the transformation of verb phrase rules.

4.2.4.2 Complex Subcategorization

By far the most important comparisons are the similarities rather than the differences between the flat and the nested methods of handling VP structure. These are embodied in the progression of grammars described above. The techniques encode the same information in ways that reflect the direct isomorphisms between them. However, the nested technique for subcategorization can be extended to allow verbs to subcategorize relative to any aspect of the DAG associated with a complement, not just the category. The grammar above can be rewritten as shown below to allow arbitrary information about complements to be subcategorized for by unifying the elements of the syncat list with the whole DAG associated with the complement, not just the *cat* subDAG.

$$VP_1 \rightarrow VP_2\ \alpha_i$$

$$\begin{aligned} <VP_2\ \mathit{syncat\ first}> &= <\alpha_i> \\ <VP_1\ \mathit{syncat}> &= <VP_2\ \mathit{syncat\ rest}> \end{aligned}$$

$$VP \rightarrow V$$

$$<VP\ \mathit{syncat}> = <V\ \mathit{syncat}>$$

4.2.5 Coordinating Syntax and Semantics Revisited

We now return to our discussion of the coordination of syntactic complement structure and semantic argument structure. Our grammar so far has the complement structure of the verb recorded in the feature ***syncat*** and the semantic structure in the feature ***trans***. Since all of the information for the mapping is thus available in the lexical entry, we can perform the mapping directly by unifying the translations of the various subcategorized elements with the various argument positions. For symmetry, we add the preverbal NP to the ***syncat*** list so that it too can be unified into the translation. Our grammar becomes

$S \rightarrow NP\ VP$

$$\begin{aligned} \langle \textit{S head} \rangle &= \langle \textit{VP head} \rangle \\ \langle \textit{NP head agr} \rangle &= \langle \textit{S head agr} \rangle \\ \langle \textit{VP syncat first} \rangle &= \langle \textit{NP} \rangle \\ \langle \textit{VP syncat rest} \rangle &= \lambda \end{aligned}$$

$VP_1 \rightarrow VP_2 NP$

$$\begin{aligned} \langle \textit{VP}_1 \textit{ head} \rangle &= \langle \textit{VP}_2 \textit{ head} \rangle \\ \langle \textit{VP}_2 \textit{ syncat first} \rangle &= \langle \textit{NP} \rangle \\ \langle \textit{VP}_1 \textit{ syncat} \rangle &= \langle \textit{VP}_2 \textit{ syncat rest} \rangle \end{aligned}$$

$VP \rightarrow V$

$$\begin{aligned} \langle \textit{VP head} \rangle &= \langle \textit{V head} \rangle \\ \langle \textit{VP syncat} \rangle &= \langle \textit{V syncat} \rangle \end{aligned}$$

Uther:

$$\begin{aligned} \langle \textit{cat} \rangle &= \textit{np} \\ \langle \textit{head agr number} \rangle &= \textit{singular} \\ \langle \textit{head agr person} \rangle &= \textit{third} \\ \langle \textit{head trans ref} \rangle &= \textit{uther}' \end{aligned}$$

Arthur:

$$\begin{aligned} \langle \textit{cat} \rangle &= \textit{np} \\ \langle \textit{head agr number} \rangle &= \textit{singular} \\ \langle \textit{head agr person} \rangle &= \textit{third} \\ \langle \textit{head trans ref} \rangle &= \textit{arthur}' \end{aligned}$$

knights:

$$\begin{aligned} \langle \textit{cat} \rangle &= \textit{v} \\ \langle \textit{head agr number} \rangle &= \textit{singular} \\ \langle \textit{head agr person} \rangle &= \textit{third} \\ \langle \textit{head trans pred} \rangle &= \textit{knight}' \\ \langle \textit{head trans arg1} \rangle &= \langle \textit{syncat rest first head trans} \rangle \\ \langle \textit{head trans arg2} \rangle &= \langle \textit{syncat first head trans} \rangle \\ \langle \textit{syncat first cat} \rangle &= \textit{np} \\ \langle \textit{syncat rest first cat} \rangle &= \textit{np} \\ \langle \textit{syncat rest rest} \rangle &= \Lambda \end{aligned}$$

4.2.6 Auxiliaries

Handling auxiliary verbs is a related question. It seems that here a nested structure (as in GPSG or PSG) is relatively well agreed upon. Thus, a rule of the form

$$VP_1 \rightarrow V\ VP_2$$

$$\begin{aligned} <VP_1\ head> &= <V\ head> \\ <V\ head\ aux> &= + \\ <V\ syncat> &= <VP_2> \\ <VP_1\ syncat> &= <VP_2\ syncat> \\ <VP_1\ syncat\ rest> &= \Lambda \end{aligned}$$

would suffice to handle auxiliaries for the nested-structure grammar. Here the *syncat* of the V will require certain features to be obtained on the sibling VP (VP_2), say that its *form* feature be *nonfinite*. By making *form* a *head* feature, we guarantee that the form of a VP comes from its first auxiliary, since the auxiliary is the head of its VP ancestor (VP_1). Finally, all the complements of VP_2 must be attached before permitting auxiliaries, and the *syncat* feature—now possessing information only about the preverbal constituent—passes from lower to upper VP, that is, from VP_2 to VP_1.

Note that the verb is required to have a + value for the *aux* feature. The VP → V rule presented earlier must be augmented by the restriction that the *aux* feature be −.

4.2.7 Adverbial Modifiers and the Generalized Wasow Effect

Modifiers can be easily dealt with in the nested-structure framework by a single rule, e.g.,

$$VP_1 \rightarrow VP_2\ ADVP$$

$$<VP_1\ syncat> = <VP_2\ syncat>$$

This rule allows adverbials to occur freely among the complements of a verb, embodying the so-called *Generalized Wasow Effect*[4], which is evident in such sentences as

1) Uther gave Lancelot on Thursday a sword.

Questions about how semantics would be affected and what head features should be percolated are as yet unresolved.

---

[4]The phenomenon and its name were brought to our attention by Ivan Sag.

4.2.8 An Implementation Notation for Grammar Writing

The PATR-II formalism can be viewed as a formal language for defining natural-language grammars. Unfortunately, as with many formal languages, the notation we have described so far is somewhat clumsy and verbose. Furthermore, there is no way to capture certain generalizations about the lexicon that a user might want to encode. We shall now describe a specific implementation of a natural-language-processing system whose underlying formalism is PATR-II and whose users are able to tailor the notation to their intended use of the formalism. As before, the intention is not to impose any particular usage, but to allow users to design their own mode of operation. The utilization of the formalism that has been described in this section has benefited from the notation, but so would many other implementations based on different strategies. The current PATR-II implementation supports the notation. Without it, our lexicon would be much more redundant.

4.2.8.1 Templates

Lexical items often share a great deal of structure because of their intended application or similarities in the way they function. We would like to define *template* DAGs that can be combined to form the lexical items in such cases. For instance, many verbs in English will share certain subcategorization information, such as a single noun-phrase complement that comprises the second argument of the predicate/argument structure. We can define a template called *Transitive* to encode this information:

Let *Transitive* be

$$\begin{aligned} \langle \textit{syncat first cat}\rangle &= \textit{np} \\ \langle \textit{syncat rest first cat}\rangle &= \textit{np} \\ \langle \textit{syncat rest rest}\rangle &= \Lambda \\ \langle \textit{head trans arg1}\rangle &= \langle \textit{syncat rest first head trans}\rangle \\ \langle \textit{head trans arg2}\rangle &= \langle \textit{syncat first head trans}\rangle \\ \langle \textit{head aux}\rangle &= \textit{false} \end{aligned}$$

Templates for *V* and *3sing*, respectively, can encode the fact that the word is a verb and that it is in the third person singular form.

Let *V* be

<cat> = V

Let *3sing* be

<head agr number> = singular
<head agr person> = third

The lexical entry for *knights* then becomes

knights:

V Transitive 3sing
<head trans pred> = knight′

4.2.8.2 Path Abbreviations

Like DAGs, path specifications can be abbreviated by using the same syntax. For example, the path abbreviation

Let *Pred* be

<head trans pred>

allows the same lexical item, *knights*, to be encoded

knights:

V Transitive 3sing
Pred = knight′

In summary, the use of templates and path abbreviations to tailor an implementation of PATR-II to a particular intended usage allows the grammar writer to capture the generalizations pertinent to that usage, at the same time facilitating the task of grammar writing and debugging by partitioning the grammar writing process into modules. Lexical rules provide a similar tool for accomplishing these objectives.

4.2.8.3 Lexical Rules

To encode the relationships among various lexical items—for instance, between the passive and active forms of a verb—we need a notion of a *lexical rule*. A lexical rule takes as input a single DAG and generates an output DAG by means of unifications. These DAGs are denoted by the metavariables *in* and *out*, respectively.

As an example, we first discuss the active-passive dichotomy. Rather than generate the active from the passive or *vice versa*, we generate both of them from a protoentry for the verb whose *syncat* feature is exactly like the *syncats* presented previously, except that the final node is not marked with a Λ and an arc *<syncat tail>* is added pointing to the final node in the *syncat* list. The *Transitive* template now looks like the following:

Let *Transitive* be

$$\begin{aligned}
\langle \textit{syncat first cat} \rangle &= \textit{np} \\
\langle \textit{syncat rest first cat} \rangle &= \textit{np} \\
\langle \textit{syncat rest rest} \rangle &= \langle \textit{syncat tail} \rangle \\
\langle \textit{head trans arg1} \rangle &= \langle \textit{syncat first head trans} \rangle \\
\langle \textit{head trans arg2} \rangle &= \langle \textit{syncat first rest head trans} \rangle \\
\langle \textit{head aux} \rangle &= \textit{false}
\end{aligned}$$

A lexical rule *active* is now defined to take a protoentry whose *syncat* was generated in this form as input and to generate an entry whose *<syncat tail>* is Λ. *Passive*, on the other hand, takes the same protoentry and moves the first element of the *syncat* list to the end of the list (the tail), thus making it a postverbal complement and making the previous leftmost postverbal complement the subject. Formally, expressed, we have

Define *Active* as

$$\begin{aligned}
\langle \textit{out cat} \rangle &= \langle \textit{in cat} \rangle \\
\langle \textit{out head} \rangle &= \langle \textit{in head} \rangle \\
\langle \textit{out head voice} \rangle &= \textit{active} \\
\langle \textit{out syncat} \rangle &= \langle \textit{in syncat} \rangle \\
\langle \textit{out syncat tail} \rangle &= \Lambda
\end{aligned}$$

Define *AgentlessPassive* as

$$\begin{aligned}
\langle \textit{out cat} \rangle &= \langle \textit{in cat} \rangle \\
\langle \textit{out head} \rangle &= \langle \textit{in head} \rangle \\
\langle \textit{out head voice} \rangle &= \textit{passive} \\
\langle \textit{out syncat} \rangle &= \langle \textit{in syncat rest} \rangle \\
\langle \textit{out syncat tail} \rangle &= \Lambda
\end{aligned}$$

Define *AgentivePassive* as

$$\begin{aligned}
\langle \textit{out cat} \rangle &= \langle \textit{in cat} \rangle \\
\langle \textit{out head} \rangle &= \langle \textit{in head} \rangle
\end{aligned}$$

<out head voice> = *passive*
<out syncat> = *<in syncat rest>*
<out syncat tail first cat> = *pp*
<out syncat tail first lex> = *by*
<out syncat tail first head trans> = *<in syncat first head trans>*

The operation of the three lexical rules on the protoentry for the verb *knight* is shown as an example. First the protoentry:

```
[cat: v
 head: [aux: false
        form: nonfinite
        trans: [pred: knight
                arg1: <f1134>
                      []
                arg2: <f1138>
                      []]]
 syncat: [first: [cat: np
                  head: [trans: <f1134>]]
          rest: [first: [cat: np
                         head: [trans: <f1138>]]
                 rest: <f1140>]
          tail: <f1140>]]
```

The active form is

```
[cat: v
 head: [aux: false
        form: nonfinite
        voice: active
        trans: [pred: knight
                arg1: <f1134>
                      []
                arg2: <f1138>
                      []]]
 syncat: [first: [cat: np
                  head: [trans: <f1134>]]
          rest: [first: [cat: np
                         head: [trans: <f1138>]]
                 rest: <f1140>
                       Λ]
          tail: <f1140>]]
```

The agentless passive form is

```
[cat: v
 head: [aux: false
```

```
              form: nonfinite
              voice: passive
              trans: [pred: knight
                      arg1: []
                      arg2: <f1138>
                            []]]
       syncat: [first: [cat: np
                        head: [trans: <f1138>]]
                        rest: <f1140>
                              Λ]
                tail: <f1140>]]
```

Finally, the agentive passive:

```
      [cat: v
       head: [aux: false
              form: nonfinite
              voice: passive
              trans: [pred: knight
                      arg1: <f1134>
                            []
                      arg2: <f1138>
                            []]]
       syncat: [first: [cat: np
                               head: [trans: <f1138>]]
                rest: [first: [cat: pp
                               lex: by
                               head: [trans: <f1134>]]
                       rest: <f1140>
                             Λ]
                tail: <f1140>]]
```

4.2.8.4 Advantages of the Notation

It has been mentioned already that the notation we have introduced, and which is used throughout the lexicon, allows convenient abbreviations. Let us exemplify this claim by presenting a lexical entry in both its full and abbreviated forms. Here is an entry for the verb *seem*:

```
seem        V - TakesIntransSbar  Monadic  Extrapos
              - TakesInf   RaisingtoS;
```

This entry collapses two verb entries for *seem*. Both entries share the category, i.e., both forms are verbs. The dashes indicate the start of each subentry. The following two sentences provide examples of the different syntactic environments that distinguish the two forms:

It seems that Uther sleeps.
Uther seems to sleep.

What follows are the definitions for the templates that are contained in the complex entry.

Let *be* be

```
                  <cat> = v
             <head aux> = false
       <head trans pred> = <sense>
```

Let *TakesIntransSbar* be

```
       <syncat first cat> = sbar
           <syncat tail> = <syncat rest>
```

Let *Monadic* be

```
       <head trans arg1> = <syncat first head trans>
```

Let *TakesInf* be

```
              <syncat first cat> = np
         <syncat rest first cat> = vp
<syncat rest first head form> = infinitival
              <syncat rest rest> = <syncat tail>
```

Let *RaisingtoS* be

```
          <head trans arg1> = <syncat rest first head trans>
<syncat rest first syncat first>= <syncat first>
```

The first subentry also contains a name of a lexical rule, Extrapos. Here is the rule:

Define *Extrapos* as

```
              <out cat> = <in cat>
             <out head> = <in head>
         <out head aux> = false
     <out head agr per> = p3
  <out syncat first cat> = np
  <out syncat first lex> = it
      <out syncat rest> = <in syncat rest>
```

$$\begin{aligned} \textit{<out syncat tail>} &= \textit{<in syncat tail rest>} \\ \textit{<in syncat tail first>} &= \textit{<in syncat first>} \end{aligned}$$

Without the notational tools introduced in this section we would have to write the following two verb entries for the two forms of *seem*:

seem₁:

$$\begin{aligned} \textit{<cat>} &= \textit{v} \\ \textit{<head aux>} &= \textit{false} \\ \textit{<head trans pred>} &= \textit{<sense>} \\ \textit{<syncat first cat>} &= \textit{np} \\ \textit{<syncat tail>} &= \textit{<syncat rest rest>} \\ \textit{<head trans arg1>} &= \textit{<syncat rest first head trans>} \\ \textit{<syncat first lex>} &= \textit{it} \\ \textit{<syncat rest first cat>} &= \textit{sbar} \\ \textit{<head agr per>} &= \textit{p3} \end{aligned}$$

seem₂:

$$\begin{aligned} \textit{<cat>} &= \textit{v} \\ \textit{<head aux>} &= \textit{false} \\ \textit{<head trans pred>} &= \textit{<sense>} \\ \textit{<syncat first cat>} &= \textit{np} \\ \textit{<syncat rest first cat>} &= \textit{vp} \\ \textit{<syncat rest first head form>} &= \textit{infinitival} \\ \textit{<syncat rest rest>} &= \textit{<syncat tail>} \\ \textit{<head trans arg1>} &= \textit{<syncat rest first head trans>} \\ \textit{<syncat rest first syncat first>} &= \textit{<syncat first>} \end{aligned}$$

In fact, these entries are the structures that are built from the "short" lexical entry when the word *seem* is encountered in an input sentence.

But our notation does not only allow convenient abbreviations; it also plays an important role in the linguist's use of the formalism. The actual format of the rules and lexical entries written by the linguist can be detached from the formalism. The grammars look more like those to which he is accustomed. Moreover, and perhaps most importantly, grammar writers can use the notational tools to express generalizations they could not state in the "pure" unification notation of the formalism. The fact that the DAGs associated with a syntactically motivated verb class like raising-to-object share some structure can be expressed in a nonredundant way, even if the amount of structure held im common cannot be encoded in a single unification statement. The linguistic observation that all English modals are finite is expressed by including the template Finite in the definition of the template Modal.

The definition of the notational tools can also be used by the grammar writer to induce constraints upon the form and power of the grammar. One could reserve lexical rules for certain types of regularities such as relation-changing rules. It is quite conceivable that, at some point, the rules and lexical entries of our grammars will contain nothing but justified abbreviations of the kind introduced above.

4.2.9 Future Research on Uses of the Formalism

Clearly, the coverage of the grammar needs further expansion. But there are also more basic questions that require closer attention than how to handle other grammatical phenomena. The linguistic status of templates and lexical rules needs to be determined. One could adopt a simple view and use lexical rules every time the power of pure unification with a template does not suffice, i.e., whenever changing to the graph structure of lexical entries requires more than the simple addition of arcs and nodes. It would be more gratifying, though, if one had a clearer correspondence between the use of notational tools, on the one hand, and classes of linguistic regularities, on the other.

Another set of problems arises with the planned integration of non-truth-conditional and pragmatic information. If the truth-conditional part of the semantics of a phrase is incorporated in the DAG, there is no obvious reason to exclude the presuppositional elements of its meaning. The details of such a solution as well as of its interaction with discourse representations need to be worked out in the course of further research.

4.3. The Current PATR-II Implementation

4.3.1 Overview

The development of the PATR-II implementation took place on the SRI-AI DEC 2060 time-sharing system operating under TOPS-20. The original implementation is written in INTERLISP-10. In order to integrate PATR-II with the other components of KLAUS, the prototype INTERLISP version needed to be transported to a LISP machine. This version now runs on a SYMBOLICS 3600 in ZETALISP. A third version of PATR-II was programmed in PROLOG. This implementation does not include all the components of the prototype. It served mainly as a testbed for a structure-sharing unification algorithm.

The prototype implementation has five major program components: a set of top-level functions; a component for building and handling the internal lexicon; the morphology component; a context-free parser; a set of functions for structure unification. The grammar consists of a set of syntactic rules, a lexicon for basic word forms, a set of affix lexicons, definitions of lexical rules, templates and path names, and a set of finite state automata representing the morphophonemic regularities of English.

4.3.2 Implementation of the Basic Formalism

4.3.2.1 Top Level and User Interface

The top-level component starts the program, initializes global variables, sets user privileges, and runs the user interface. The main function for the user interface is COMMANDS. It will prompt the user with "command or sentence to parse." At this level, the user can give commands that load and clear grammars, parse sentences, debug, trace, and edit the grammar and lexicon, save any desired versions of the grammar, or save the whole system.

If the user input is enclosed in parentheses, the expression will be evaluated as an INTERLISP S-expression. If a sentence is given instead of a command, PATR-II will attempt to parse it turning control over to the parser for this purpose. The parser activates lexical lookup, morphological analysis, phrase structure building, and graph unification. If parses are found, the corresponding semantic translations will be printed out.

4.3.2.2 The Lexicon Functions

PATR-II actually has several lexicons: a stem (or root) lexicon and several small affix lexicons. Lexicons written in the notation described in Section 4.2.8 are stored internally as letter trees. The lexical information of an entry in these trees is associated with its last letter. The trees are used as discrimination networks for lexical lookup. There are functions that add, delete, display, and change entries. Other functions build new internal lexicons from inputted lexicon files or write out letter trees in the linguistic format.

4.3.2.3 Morphological Analysis

The lexicon for a language-processing system should not have to list the full morphological paradigm for each entry when there are many indications of the productivity of morphological rules for such processes as plural formation, conjugation, and English genitive inflection. On the other hand, the regularities that govern these processes are quite different from those entailed in syntactic processes and, moreever, it is impossible to separate morphological from phonological rules. Therefore, one often speaks about the morphophonemic component of a grammar. The design of PATR-II takes the special status of morphophonemic processes into account by assigning them to a separate component of the system: the morphological analyzer.

Our morphological analyzer is based on a recent implementation of Kimmo Koskenniemi's "bi-level model" for morphological analysis and synthesis [64]. This implementation was developed in INTERLISP as a course project at the University of Texas under the direction of Lauri Karttunen [57].

Two-level rules do not describe transformations of segment sequences in the same way as do rules of generative phonology. They are simply descriptions of correspondences between lexical and surface forms. In this respect the model resembles old-fashioned structural phonology, although it also differs from the letter in several important ways. Just as in structural phonology, in the two-level model there are no rule interactions, no relationships such as the bleeding or feeding that result from the sequential application of rules, so that subsequent rules apply to the output of earlier ones.

Like the other parts of the PATR-II processor, the morphological component is language-independent.

Morphological rules are represented in the processor as automata—more specifically, as finite-state transducers. There is a one-to-one correspondence between the rules and the automata. The idea of compiling rules into finite-state machines comes originally from Martin Kay and Ronald Kaplan [55]. In addition to the functions that analyze morphological forms by running the finite-state automata there are functions that compile and merge these automata from sets of phonological rules.

4.3.2.4 The Parser

The parser of the INTERLISP prototype PATR-II is a context-free, bottom-up chart parser without lookahead. It was inspired by the Bear-Karttunen PSG parser , which in turn is based on Dan Chester's implementation of the Cocke, Kasami, Younger algorithm (refer to [6] for a description of the parser and algorithm).

Before a new constituent is added to the chart, the DAGs of parent and children nodes are selectively unified by the graph unification component according to the unifications listed in the body of the applied rule. The completed edges of the chart of the PATR-II parser include pointers to the DAGs associated with the nodes.

The treatment of long-distance gap-filler dependencies is based on the opinion that the phenomenon is so general that the processor and not the grammar should be responsible for introducing and percolating gaps. Consequently, no grammar rules have to be duplicated to account for gap production. The current solution resembles the one in the PSG parser: the parser simply "assumes" a trace between every two adjacent words in the input. These traces can stand for NPs or PPs. Their agreement features are carried up the tree and are unified in the end with the filler's agreement features.

We intend to replace the parser with a "smarter," more predictive one later, that will recognize potential gaps only at places where they can really occur. We also want to investigate how a phrase-linking solution of the type proposed by Peters and Ritchie [100] could be implemented in the PATR-II formalism.

4.3.2.5 Graph Unification

As described before, a PATR-II grammar is a set of context-free rules annoted with DAG unifications. It is useful to approach the problem of constructing a parser for PATR-II by determining which extensions should be made of a context-free parser to enforce the constraints specified by unifications. However, some parsing strategies that are reasonable for context-free grammars are not applicable or are just too inefficient for the extended parser. This is especially the case with parsers that require the context-free grammar to be rewritten into some normal form, because, in general, an annoted grammar cannot be rewritten this way.

The prototype PATR-II parser is a pure bottom-up context-free parser that applies all unifications associated with a rule when the latter is used to build a new phrase (parent) from its subphrases (children). When unifications are applied, both the parent phrase and the children may become more specified (more "instantiated"). Because of local or global ambiguities in the grammar, a given phrase may appear as the child of more than one parent phrase by virtue of rule applications that instantiate the child in different ways. For the parser to work properly, these alternative instantiations of the DAG associated with a phrase must be segregated. The prototype achieves this segregation by copying all the child phrases and their DAGs before trying to apply a rule (even if the rule application may eventually fail because of contradictory unifications or values).

The copying method used in the prototype is easy to implement, which is the main reason it was chosen. However, the wholesale copying of DAGs with each rule application requires far more space (and time) than the "structure sharing" method we are now considering. A further problem with the prototype parser is that the pure bottom-up strategy has difficulty in dealing with missing constituents (gaps) in a general manner.

The structure-sharing method of DAG representation and unification is closely modeled on the technique of the same name used in automatic theorem provers. This connection between

PATR-II parsers and theorem provers is more than coincidental, as it derives from the very close inherent relationship between PATR-II grammars and first-order theories.

Using structure sharing, the DAG associated with a phrase is represented by a pair of items: its "skeleton"—a DAG derived by compile-time application of all the unifications of a single rule; its "index," a number identifying the particular rule application that created this DAG. The index of a DAG is used to tag records ("bindings") that describe additions made to the DAG through unification. Bindings are stored in "binding environments." Although each alternative partial analysis of the input has its own binding environment, most of these environments share information because they have been derived (through alternative rule applications) from earlier partial analyses.

The standard structure-sharing technique, invented by Boyer and Moore [11], requires an amount of searching for the bindings of a DAG that, at worst, can be proportional to the size of the preceding analysis. Instead of this scheme, we have in our experiments with structure sharing adopted a "logarithmic tree" representation of binding environments and a parsing strategy that make binding lookup at worst logarithmic with respect to the size of the analysis. The parsing strategy used, which is a variation of the Earley parsing technique, has the further advantage of allowing gap-introducing rules with full generality. However, to achieve the logarithmic time bound forces us to copy new complete phrases as they are created, although partial analyses are still fully structure-shared. Since the trade-offs between this method and the standard structure-sharing one are difficult to identify theoretically, we plan to implement another version of the structure sharing PATR-II parser, using the standard Boyer and Moore method.

The Prolog implementation of PATR-II is based on an experimental structure-sharing parser of the kind described above.

4.3.3 PROLOG Implementation

Besides the INTERLISP and the LISP machine implementations of PATR-II, there exists also a PROLOG implementation of the basic formalism on the DEC-2060. It is based on the experimental structure-sharing PATR-II parser described in Section 4.3.2.5. The Prolog program has run successfully with various PATR-II grammars, with an efficiency similar to that of the copying parser. Prolog has been useful for rapid "throw away" testing of alternative parsing mechanisms. The advantages of a structure-sharing parser are expected to dominate performance for larger grammars than our current ones, at which point it will become worthwhile to reimplement the parser using more efficient low-level coding on the LISP machines.

4.3.4 LISP Machine Implementation

For the integration of PATR-II with the other components of KLAUS, the prototype PATR-II implementation that had been developed in INTERLISP on a DEC 2060 had to be transported to a Symbolics 3600 LISP machine.

The transfer of PATR to the 3600 was done in such a fashion that further development could be done on the 2060 and, at the same time, make it relatively easy to retransport to the 3600. An initial effort to translate the INTERLISP-10 code into ZETALISP code directly by using the INTERLISP TRANSOR translator revealed a number of problems.

An alternative method was tried. Symbolics offers an INTERLISP Compatibility Package (ILCP) consisting of a translator that runs under INTERLISP and a run time package that runs on the 3600. The translator on the INTERLISP end mainly checks for upper/lower-case problems, handles comments, and other syntactic features of INTERLISP. The run time package on the 3600 provides a simulated INTERLISP environment. That is, many of the INTERLISP functions are defined to work as they do in INTERLISP. For instance, MAP takes its arguments in the order used by INTERLISP rather than the opposite order used by ZETALISP.

The disadvantage of this method is that the ILCP is a rather large set of software that is still being developed. It was necessary to rewrite all the INTERLISP I/O functions, as the supplied definitions did not cover PATR's particular usage of those functions. At present, this software package must be loaded each time the PATR code is loaded. However, it was decided that running with the ILCP was easier and more reliable than using a special translator for the PATR code.

Some functions required by PATR, e.g., ASKUSER, were not available. This function was coded directly in ZETALISP to use normal mouse selection when possible.

When the PATR code was run on the 3600 under the ILCP, several coding problems were discovered. Most of these were avoidable (i.e., taking the CAR of an atom) and a patch was made on both the INTERLISP and ZETALISP versions of PATR.

The only serious difference between these two versions is in the treatment of case distinction. ZETALISP is indiscriminate; it translates all normal input into upper-case. INTERLISP-10, on the other hand, leaves all input in its original case.

PATR made extensive use of the lower/upper-case distinction, but, fortunately, most of this selectivity was aesthetic rather than essential. We were able to modify the code so that, in almost all cases, the program works regardless of whether or not lower and upper case are merged. In one or two places, where the difference could not be compensated for by coding, the code had to be hand-patched when translated from INTERLISP-10 to ZETALISP.

The above describe the differences in the running code of the two systems. Other differences are found in the user interface. Some of the utilities provided by the INTERLISP-10 system are moot on the ZETALISP system, e.g., EXIT and SAVE. Others, such as EDIT, were not directly available on the ZETALISP system. EDIT was coded to be as much like the INTERLISP-10 system as possible. DRIBBLE was a feature that could not be provided without a substantial coding effort.

The section of code for the user interface needed to be redone. As a result, a menu of the available commands is now permanently displayed on the screen. To choose a command, the user simply points the mouse and clicks one of the buttons (usually the left) on the mouse. In a few cases, clicking the middle or right button provides for different options of the basic command. For instance, clicking right on FASTLOAD allows the user to specify the system name first.

As PATR outputs text to the screen, it specifies to the 3600 for certain items of text what type of item is being displayed. For instance, when it outputs the name of a DAG, it informs the 3600 that it is outputting a DAG. If PATR later asks for the name of a DAG, the user may either type in a DAG or point the mouse to one of the printed DAG names. When he points to this DAG, a box appears around it, indicating that it is a possible answer. If he then clicks a button on the mouse, that DAG name is inputted.

Almost all of these user interface changes are in the top-level routine that takes commands from the user. Therefore, it can be just loaded in place of the INTERLISP-10 command interface. Any new commands can be easily added. A few changes were necessary in the body of the system where output is done so that, for instance, the 3600 is told when a DAG name is printed. These changes were also added to the INTERLISP-10 version.

The utilization of the display and user interface features of the 3600 have created a superior working environment. The menu-driven top-level functions, together with the multi-window display improve grammar and program development. New debugging and editing facilities that utilize the available ZETALISP function packages are still being added to the system.

4.3.5 Future Research on the Implementation

Among the related projects we want to undertake next are the implementation of the structure-sharing unification algorithm on the LISP machine, the development of a phrase structure parser with more predictive power, and a phrase-linking solution to unbounded dependencies [100].

4.4. Conclusion

Major parts of our implementation are a grammar formalism and an implementation notation designed to serve as a "programming language for linguists." That is, it is a powerful grammar-writing system that allows the encoding of many analyses of linguistic phenomena.

In the sense that the formalism does not attempt to characterize all and only the grammars of natural languages (though a more constrained theory might use the formalism as its "semantics" so to speak), it does not embody a linguistic theory. Instead, it is a tool linguists can use to express linguistic analyses formally; its implementation is a tool for testing such expressions.

The formalism and the notation for grammar writing proved to be adequate and convenient devices for writing grammars that cover the grammatical phenomena we have dealt with so far. The notation has also shown itself to be useful as a conceptual aid in the formulation of linguistic-research problems.

Our modular implementation, consisting of a top level, a parser, a unification component, and a morphological analyzer, makes it easy to replace any individual component.

The current implementation was designed as a research tool. This means that the advantages modularity and the convenience of modifying grammars as well as implementation had priority over efficiency. Nevertheless, the process of parsing and translating sentences of different degrees of complexity is performed at reasonable speed.

Appendix A. A Formal Definition of the Formalism

Definition 4.1. DAG:

A DAG defined over a finite set of labels A is either

- an atomic label $l \in A$, or
- a possibly empty set s of pairs $< l, v >$ where $l \in A$ and v is a DAG and s does not *cover* s. (Covering is defined recursively as follows: for all $< l, v > \in s$, s covers v and s covers anything covered by v. The atomic label l covers only itself.)

l is called the *attribute* or *feature* and v the *value* of the attribute.

Definition 4.2. Path:

A *path* is a sequence $< n, l_1, \ldots, l_m >$ (hereafter notated without commas so as to avoid confusion with other sequences) where n is a DAG and the l_i are atomic labels. Such a path *denotes* the node n_m where $< l_i, n_i > \in n_{i-1}$.

Definition 4.3. Grammar:

A PATR-II *grammar* is a sextuple $< N, T, A, R, L, S >$ where

- N is a finite nonempty set of nonterminals,
- T is a nonempty set of terminals,
- A is a finite set of labels, (usually a superset of $N \cup T$),
- R is a finite set of grammar rules (see below),
- L is a relation in $T \times D$, where D is the set of DAGs definable over A, and
- $S \in D$ is the start DAG.

Definition 4.4. Grammar rule:

A grammar rule has two parts:

- a context-free phrase structure rule with uniquely identified nonterminals, notated, e.g., $VP \rightarrow V\ NP_1\ NP_2$
- a set of unifications, notated as $m = n$ where m and n are DAGs or path specifications with nonterminal labels instead of DAGs as the first elements, e.g., $< VP\ l_1\ l_2 > = < V\ l_3 >$.

Definition 4.5. Admissibility:

A rule $l_0 \rightarrow l_1 \cdots l_m$ with unifications U *admits* a sequence of DAGs $< n_0, n_1, \ldots, n_m >$ iff

- if $l_i \in T$, then $< l_i, n_i > \in L$, and
- if $l_i \in N$, then the path $< n_i\ cat >$ denotes l_i (minus any subscripts), and
- for all $p_1 = p_2$ in U the node denoted by p_1 is the same as that denoted by p_2. A path $< l_i\ k_1 \cdots k_q >$ denotes a node n if and only if the path $< n_i\ k_1 \cdots k_q >$ denotes n.

Definition 4.6. Derivation:

A DAG n_0 *derives* a sequence of DAGs $< n_1, \ldots, n_m >$ if there is a rule $r \in R$ such that r admits $< n_0, n_1, \ldots, n_m >$. This is notated $n_0 \Rightarrow n_1 \cdots n_m$. The symmetric transitive closure of $\Rightarrow$ is notated $\Rightarrow^*$.

Definition 4.7. Language:

The *language* of a PATR-II grammar $G = < N, T, R, L, S >$ is the set $\{w \in T^* \mid S \Rightarrow^* w\}$

Definition 4.8. Unification:

The *unification* of two DAGs n_1 and n_2 is a DAG n where

- if $n_1 = n_2$, then $n = n_1$,
- if n_1 is atomic and $n_2 = \{\}$, then $n = n_1$, and similarly with n_1 and n_2 interchanged,
- if neither n_1 nor n_2 is atomic, then for all l such that $< l, v_1 > \in n_1$, and $< l, v_2 > \in n_2$, $< l, unify(v_1, v_2) > \in n$ and for all l such that $< l, v > \in (n_1 \cup n_2) - (n_1 \cap n_2)$, $< l, v > \in n$.

Appendix B. Some Grammars for Hard Languages

The following grammar accepts the non-context-free language $a^n b^n c^n$:

$S \rightarrow As\ Bs\ Cs$

$$\begin{aligned} <As> &= <Bs> \\ <Bs> &= <Cs> \end{aligned}$$

$As_1 \rightarrow As_2\ A$

$$<As_1\ succ> = <As_2>$$

$As \rightarrow \epsilon$

$$<num> = 0$$

$Bs_1 \rightarrow Bs_2\ B$

$$<Bs_1\ succ> = <Bs_2>$$

$Bs \rightarrow \epsilon$

$$<Bs> = 0$$

$Cs_1 \rightarrow Cs_2\ C$

$$<Cs_1\ succ> = <Cs_2>$$

$Cs \rightarrow \epsilon$

$$<Cs> = 0$$

a:

$$<cat> = a$$

b:

$$<cat> = b$$

c:

$$<cat> = c$$

The following grammar accepts the non-indexed language $a^{2^{2^n}}$:

$S \rightarrow A$

$$\begin{aligned} <S\ m> &= <A\ m> \\ <A\ n> &= 0 \\ <A\ p> &= <A\ q> \end{aligned}$$

$A \rightarrow B$

$$\begin{aligned}
<A\ m> &= 0 \\
<A\ n> &= <A\ p\ succ> \\
<A\ q> &= <B\ m>
\end{aligned}$$

$A_1 \rightarrow A_2\ A_3$

$$\begin{aligned}
<A_1\ m\ succ> &= <A_2\ m> \\
<A_1\ n> &= <A_2\ n> \\
<A_1\ q\ succ> &= <A_2\ q> \\
<A_1\ m\ succ> &= <A_3\ m> \\
<A_1\ p> &= <A_3\ p> \\
<A_1\ q\ succ> &= <A_3\ q> \\
<A_2\ p> &= <A_3\ p>
\end{aligned}$$

a:

$$\begin{aligned}
<cat> &= B \\
<m> &= 0
\end{aligned}$$

$B_1 \rightarrow B_2\ B_3$

$$\begin{aligned}
<B_1\ m\ succ> &= <B_2\ m> \\
<B_1\ m\ succ> &= <B_3\ m>
\end{aligned}$$

Appendix C. Conversion to Normal-Form PATR-II Grammars

In Section 4.2.4.1 we state that any context-free grammar can be converted to a PATR-II grammar with only one nonunary rule. The construction is as follows: Given a context-free grammar $< N, T, R, S >$, we construct a PATR-II grammar with the following rules:

$$S'' \rightarrow S'$$

$$<S'\,syncat> \;=\; \lambda$$

$$S' \rightarrow S_1\ S_2$$

$$\begin{aligned} <S'\,syncat> &= <S_1\ syncat\ rest> \\ <S\ syncat\ first> &= <S\ cat> \end{aligned}$$

For every rule $\alpha \rightarrow \beta_1 \cdots \beta_n \in R$, add the rule

$$S \rightarrow \alpha$$

$$\begin{aligned} <S\ syncat\ first> &= \beta_1 \\ <S\ syncat\ rest\ first> &= \beta_2 \end{aligned}$$

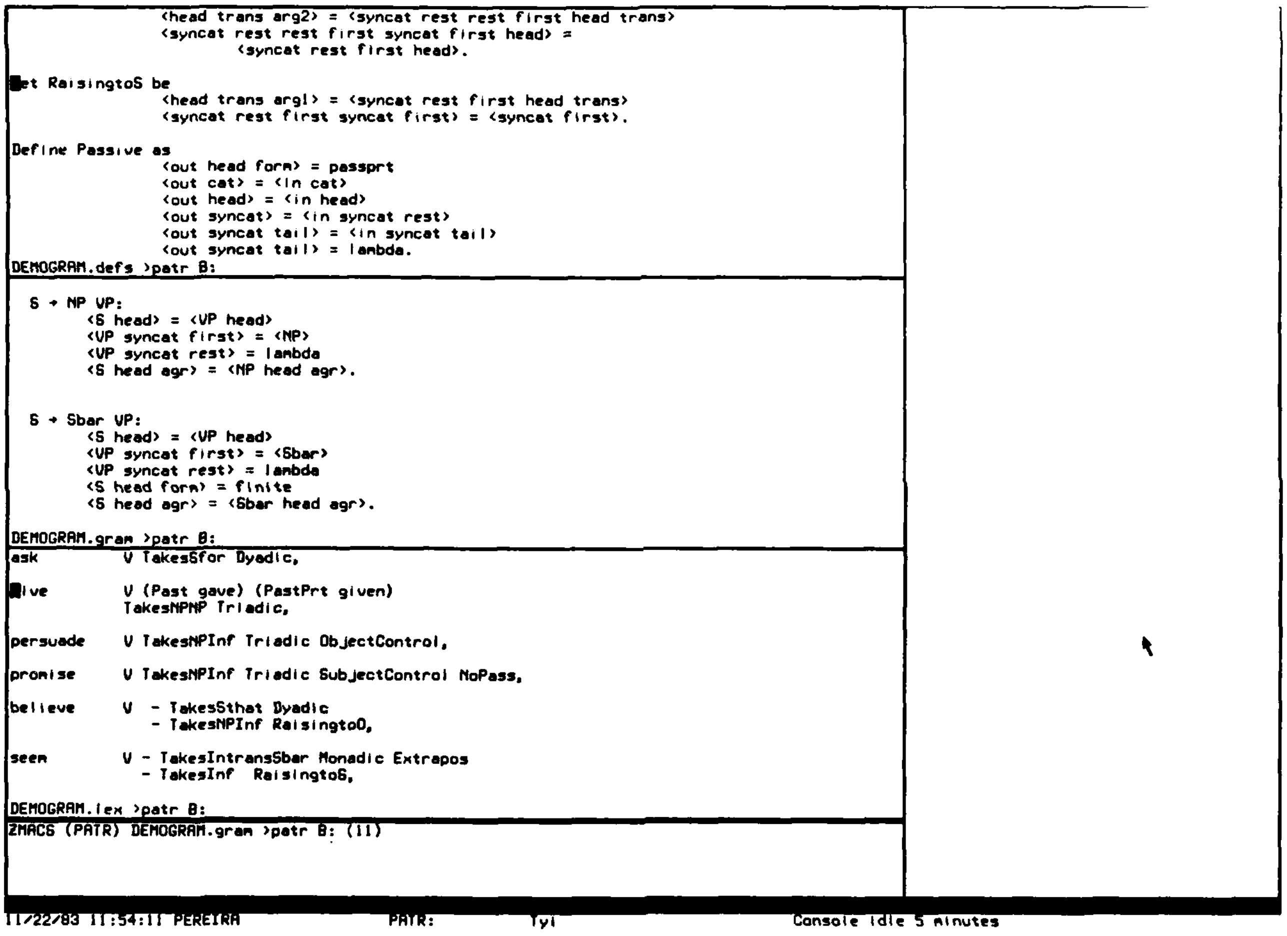
etc.,

and, for every β_i, add the rule

$$S \rightarrow \beta_i$$

Appendix D. Sample Rules of the PATR-II Grammar

The LISP machine screen below displays three editing windows with samples of definitions (one template and one lexical rule) in the upper window, of syntactic rules in the middle window, and of lexical entries in the lower window.

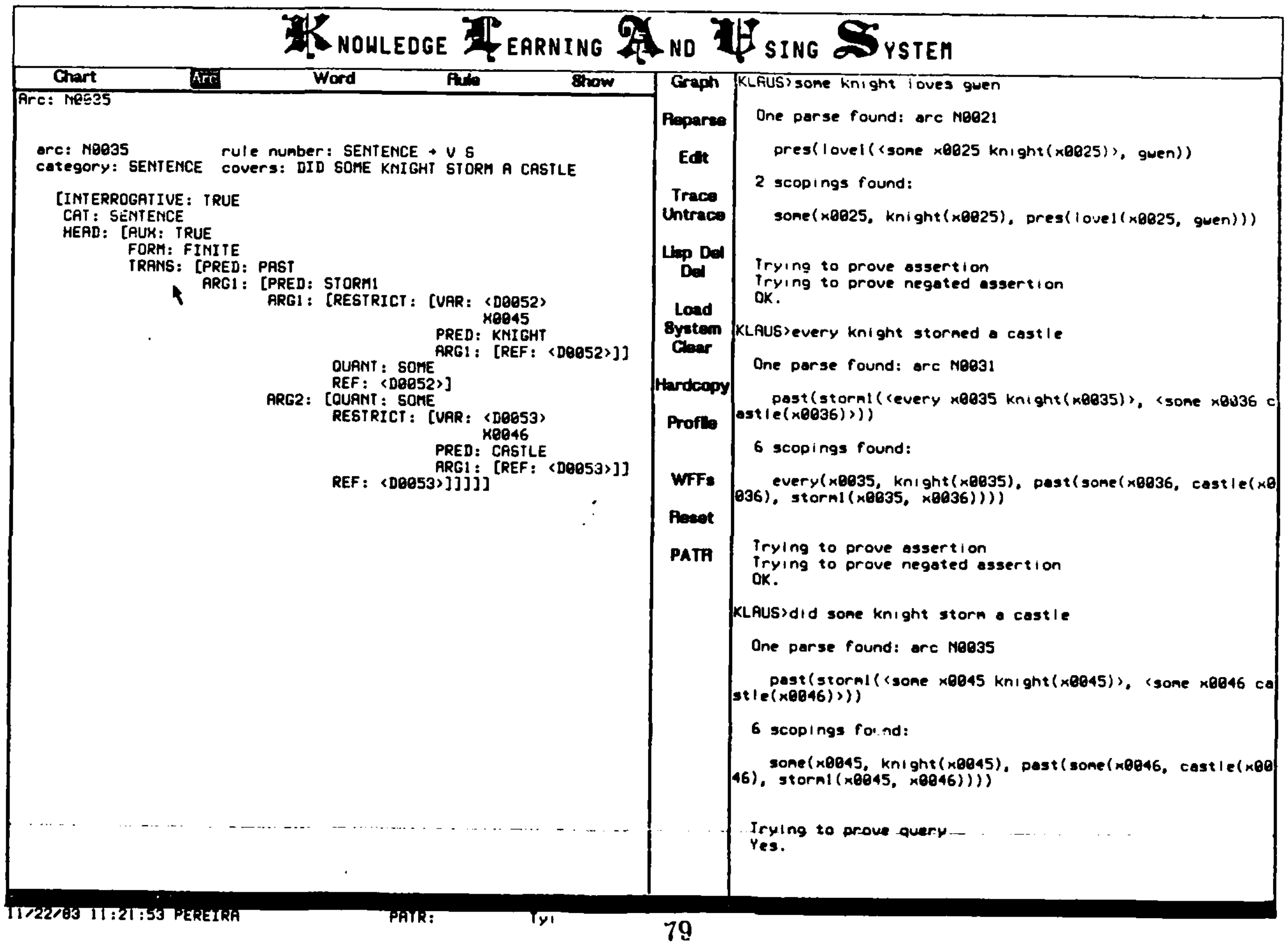


```
                <head trans arg2> = <syncat rest rest first head trans>
                <syncat rest rest first syncat first head> =
                        <syncat rest first head>.

█et RaisingtoS be
                <head trans arg1> = <syncat rest first head trans>
                <syncat rest first syncat first> = <syncat first>.

Define Passive as
                <out head form> = passprt
                <out cat> = <in cat>
                <out head> = <in head>
                <out syncat> = <in syncat rest>
                <out syncat tail> = <in syncat tail>
                <out syncat tail> = lambda.
DEMOGRAM.defs >patr B:

  S → NP VP:
        <S head> = <VP head>
        <VP syncat first> = <NP>
        <VP syncat rest> = lambda
        <S head agr> = <NP head agr>.


  S → Sbar VP:
        <S head> = <VP head>
        <VP syncat first> = <Sbar>
        <VP syncat rest> = lambda
        <S head form> = finite
        <S head agr> = <Sbar head agr>.

DEMOGRAM.gram >patr B:
ask          V TakesSfor Dyadic,

█ive         V (Past gave) (PastPrt given)
             TakesNPNP Triadic,

persuade     V TakesNPInf Triadic ObjectControl,

promise      V TakesNPInf Triadic SubjectControl NoPass,

believe      V  - TakesSthat Dyadic
                - TakesNPInf RaisingtoO,

seem         V - TakesIntransSbar Monadic Extrapos
               - TakesInf  RaisingtoS,

DEMOGRAM.lex >patr B:
ZMACS (PATR) DEMOGRAM.gram >patr B: (11)

11/22/83 11:54:11 PEREIRA          PATR:        Tyi                     Console idle 5 minutes
```

Appendix E. A Sample Dialog with PATR-II

The right-hand-side window (interaction window) of the following LISP machine freeze frame contains a three-sentence sample dialog with KLAUS. The left-hand-side window can be used for displaying the chart, DAGs, words, and rules that were built or used during the parsing of a sentence. There is a mouse-operated menu window on top of this display window. In the freeze frame, the DAG associated by the parser with the last sentence is displayed in the display window. Display window and interaction window are seperated by another menu window which represents the user's options at the top level of KLAUS.

For each of the sentences in the short dialog, the parser found exactly one parse but multiple scopings of quantifiers and tense operators. The selection of the desired scoping was performed in a temporary menu window using the mouse. Assertions as the first two sentences are accepted by KLAUS if they do not contradict with already known propositions. Possible responses to alternative questions are "YES", "NO", or "I DON'T KNOW". The answer to the third input sentence, which is an alternative question, is based on the knowledge the system gained through the first two input sentences.



```
Arc: N0035


 arc: N0035            rule number: SENTENCE → V S
 category: SENTENCE    covers: DID SOME KNIGHT STORM A CASTLE

   [INTERROGATIVE: TRUE
    CAT: SENTENCE
    HEAD: [AUX: TRUE
           FORM: FINITE
           TRANS: [PRED: PAST
                   ARG1: [PRED: STORM1
                          ARG1: [RESTRICT: [VAR: <D0052>
                                                 X0045
                                            PRED: KNIGHT
                                            ARG1: [REF: <D0052>]]
                                 QUANT: SOME
                                 REF: <D0052>]
                          ARG2: [QUANT: SOME
                                 RESTRICT: [VAR: <D0053>
                                                 X0046
                                            PRED: CASTLE
                                            ARG1: [REF: <D0053>]]
                                 REF: <D0053>]]]]]
```

```
KLAUS>some knight loves gwen

  One parse found: arc N0021

    pres(love1(<some x0025 knight(x0025)>, gwen))

  2 scopings found:

    some(x0025, knight(x0025), pres(love1(x0025, gwen)))


  Trying to prove assertion
  Trying to prove negated assertion
  OK.

KLAUS>every knight stormed a castle

  One parse found: arc N0031

    past(storm1(<every x0035 knight(x0035)>, <some x0036 c
astle(x0036)>))

  6 scopings found:

    every(x0035, knight(x0035), past(some(x0036, castle(x0
036), storm1(x0035, x0036))))


  Trying to prove assertion
  Trying to prove negated assertion
  OK.

KLAUS>did some knight storm a castle

  One parse found: arc N0035

    past(storm1(<some x0045 knight(x0045)>, <some x0046 ca
stle(x0046)>))

  6 scopings found:

    some(x0045, knight(x0045), past(some(x0046, castle(x00
46), storm1(x0045, x0046))))


  Trying to prove query
  Yes.
```

5. Sentence Disambiguation by a Shift-Reduce Parsing Technique

This section was written by Stuart Shieber.

5.1. Introduction

For natural-language-processing systems to be useful, they must assign the same interpretation to a given sentence that a native speaker would, since that is precisely the behavior users will expect. Consider, for example, the case of ambiguous sentences. Native speakers of English show definite and consistent preferences for certain readings of syntactically ambiguous sentences [59, 28, 27]. A user of a natural-language-processing system would naturally expect it to reflect the same preferences. Thus, such systems must model in some way the *linguistic performance* as well as the *linguistic competence* of the native speaker.

This idea is certainly not new in the artificial-intelligence literature. The pioneering work of Marcus [83] is perhaps the best known example of linguistic-performance modeling in AI. Starting from the hypothesis that "deterministic" parsing of English is possible, he demonstrated that certain performance constraints, e.g., the difficulty of parsing garden-path sentences, could be modeled. His claim about deterministic parsing was quite strong. Not only was the behavior of the parser required to be deterministic, but, as Marcus claimed,

> The interpreter cannot use some general rule to take a nondeterministic grammar specification and impose arbitrary constraints to convert it to a deterministic specification (unless, of course, there is a general rule which will always lead to the correct decision in such a case). [83, p. 14]

We have developed and implemented a parsing system that, given a nondeterministic grammar, forces disambiguation in just the manner Marcus rejected (i.e. through general rules); it thereby exhibits the same preference behavior that psycholinguists have attributed to native speakers of English for a certain range of ambiguities. These include structural ambiguities [28, 29, 130] and lexical preferences [27], as well as the garden-path sentences as a side effect. The parsing system is based on the shift-reduce scheduling technique of Pereira [96].

Our parsing algorithm is a slight variant of LALR(1) parsing and, as such, exhibits the three conditions postulated by Marcus for a deterministic mechanism: it is data-driven, reflects expectations, and has look-ahead. Like Marcus's parser, our parsing system is deterministic. Unlike Marcus's parser, the grammars used by ours can be ambiguous.

5.2. The Phenomena to be Modeled

The parsing system was designed to manifest preferences among structurally distinct parses of ambiguous sentences. It does this by building just one parse tree—rather than building multiple parse trees and choosing among them. Like the Marcus parsing system, ours does not do disambiguation requiring "extensive semantic processing," but, in contrast to Marcus, it does handle such phenomena as PP-attachment insofar as there exist *a priori*, preferences for one attachment over another. By *a priori* we mean preferences that are exhibited in contexts where pragmatic or *plausibility considerations do not tend to favor one reading over the other.* Rather than make such value judgments ourselves, we defer to the psycholinguistic literature (specifically [28], 29, 27]) for our examples.

The parsing system models the following phenomena:

Right Association Native speakers of English tend to prefer readings in which constituents are "attached low." For instance, in the sentence

```
Joe bought the book that I had been trying
to obtain for Susan.
```

the preferred reading is one in which the prepositional phrase "for Susan" is associated with "to obtain" rather than "bought."

Minimal Attachment On the other hand, higher attachment is preferred in certain cases such as

```
Joe bought the book for Susan.
```

in which "for Susan" modifies "the book" rather than "bought." Frazier and Fodor [28] note that these are cases in which the higher attachment includes fewer nodes in the parse tree. Our analysis is somewhat different.

Lexical Preference Ford *et al.* [27] present evidence that attachment preferences depend on lexical choice. Thus, the preferred reading for

```
The woman wanted the dress on that rack.
```

has low attachment of the PP, whereas

```
The woman positioned the dress on that rack.
```

has high attachment.

Garden-Path Sentences

Grammatical sentences such as

```
The horse raced past the barn fell.
```

seem actually to receive no parse by the native speaker until some sort of "conscious parsing" is done. Following Marcus [83], we take this to be a hard failure of the human sentence-processing mechanism.

It will be seen that all these phenomena are handled in our parser by the same general rules. The simple context-free grammar used[1] (see Appendix A) allows both parses of the ambiguous sentences as well as one for the garden-path sentences. The parser disambiguates the grammar and yields only the preferred structure. The actual output of the parsing system can be found in Appendix B.

5.3. The Parsing System

The parsing system we use is a shift-reduce parser. Shift-reduce parsers [1] are a very general class of bottom-up parsers characterized by the following architecture. They incorporate a *stack* for holding constituents built up during the parse and a *shift-reduce table* for guiding the parse. At each step in the parse, the table is used for deciding between two basic types of operations: the *shift* operation, which adds the next word in the sentence (with its preterminal category) to the top of the stack, and the *reduce* operation, which removes several elements from the top of the stack and replaces them with a new element—for instance, removing an NP and a VP from the top of the stack and replacing them with an S. The *state* of the parser is also updated in accordance with the shift-reduce table at each stage. The

[1]We make no claims as to the accuracy of the sample grammar, which is obviously a gross simplification of English syntax. Its role is merely to show that the parsing system is able to disambiguate the sentences under consideration correctly.

combination of the stack, input, and state of the parser will be called ***a configuration*** and will be notated as, for example,

stack: NP V	***input:*** Mary	***state:*** 10

where the stack contains the nonterminals NP and V, the input contains the lexical item Mary, and the parser is in state 10.

By way of example, we demonstrate the operation of the parser (using the grammar of Appendix A) on the oft-cited sentence "John loves Mary." Initially the stack is empty and no input has been consumed. The parser begins in state 0.

stack:	***input:*** John loves Mary	***state:*** 0

As elements are shifted to the stack, they are replaced by their preterminal category.[2] The shift-reduce table for the grammar of Appendix A states that in state 0, with a proper noun as the next word in the input, the appropriate action is a shift. The new configuration, therefore, is

stack: PNOUN	***input:*** loves Mary	***state:*** 4

The next operation specified is a reduction of the proper noun to a noun phrase, yielding

stack: NP	***input:*** loves Mary	***state:*** 2

The verb and second proper noun are now shifted, in accordance with the shift-reduce table, thus exhausting the input, and the proper noun is then reduced to an NP.

stack: NP V	***input:*** Mary	***state:*** 10
stack: NP V PNOUN	***input:***	***state:*** 4
stack: NP V NP	***input:***	***state:*** 14

Finally, the verb and noun phrase on the top of the stack are reduced to a VP

[2]But see Section 5.3.2 for an exception.

stack: NP VP	*input:*	*state:* 6

which is in turn reduced, together with the subject NP, to an S.

stack: S	*input:*	*state:* 1

This final configuration is an accepting configuration, since all the input has been consumed and an S derived. Thus, the sentence is grammatical according to the grammar of Appendix A, as expected.

5.3.1 Differences from the Standard LR Techniques

The shift-reduce table mentioned above is generated automatically from a context-free grammar by the standard algorithm [1]. The parsing algorithm differs, however, from the standard LALR(1) parsing algorithm in two ways. First, instead of assigning preterminal symbols to words as they are shifted, the algorithm allows the assignment to be delayed if the word is ambiguous among preterminals. When the word is used in a reduction, the appropriate preterminal is assigned.

Second, and most importantly, since true LR parsers exist only for unambiguous grammars, the normal algorithm for deriving LALR(1) shift-reduce tables yields a table that may specify conflicting actions under certain configurations. It is through the choice made from the options in a conflict that the preference behavior we desire is engendered.

5.3.2 Preterminal Delaying

One key advantage of shift-reduce parsing that is critical in our system is the fact that decisions about the structure to be assigned to a phrase are postponed as long as possible. In keeping with this general principle, we extend the algorithm to allow the assignment of a preterminal category to a lexical item to be deferred until a decision is forced upon it, so to speak, by an encompassing reduction. For instance, we would not want to decide on the preterminal category of the word "that," which can serve as either a determiner (DET) or complementizer (THAT), until some further information is available. Consider the sentences

```
That problem is important.
That problems are difficult to solve is im-
portant.
```

Instead of assigning a preterminal to "that," we leave open the possibility of assigning either DET or THAT until the first reduction that involves the word. In the first case, this reduction will be by the rule NP $\rightarrow$ DET NOM, thus forcing, once and for all, the assignment of DET as preterminal. In the second case, the DET NOM analysis is disallowed on the basis of number agreement, so that the first applicable reduction is the COMP S reduction to $\overline{\mathrm{S}}$, forcing the assignment of THAT as preterminal.

Of course, the question arises as to what state the parser goes into after shifting the lexical item "that." The answer is quite straightforward, though its interpretation *vis à vis* the determinism hypothesis is subtle. The simple answer is that the parser enters into a state corresponding to the union of the states entered upon shifting a DET and upon shifting a THAT, respectively, in much the same way as the deterministic *simulation of a nondeterministic* finite automaton enters a "union" state when faced with a nondeterministic choice. Are we then merely simulating a nondeterministic machine here? The answer is equivocal. Although the implementation acts as a simulator for a nondeterministic machine, the nondeterminism is *a priori* bounded, given a particular grammar and lexicon.[3] Thus, the nondeterminism could be traded in for a larger, albeit still finite, set of states, unlike the nondeterminism found in other parsing algorithms. Another way of looking at the situation is to note that there is no observable property of the algorithm that would distinguish the operation of the parser from a deterministic one. In some s there is no interesting difference between the limited nondeterminism of this parser and Marcus's notion of strict determinism. In fact, the implementation of Marcus's parser also embodies a bounded nondeterminism in much the same way this parser does.

[3]The boundedness comes about because only a finite amount of information is kept per state (an integer) and the nondeterminism stops at the preterminal level, so that the splitting of states does not propagate.

The property that discriminates between this parser and that of Marcus is a slightly different one, namely, the property of ***quasi-real-time operation***.[4] By quasi-real-time operation, Marcus means that there exists a maximum interval of parser operation for which no output can be generated. If the parser operates for longer than this, it must generate some output. For instance, the parser might be guaranteed to produce output (i.e., structure) at least every three words. However, because preterminal assignment can be delayed indefinitely in pathological grammars, there may exist sentences in such grammars for which arbitrary numbers of words need to be read before output can be produced. It is not clear whether this is a real disadvantage or not, and, if so, whether there are simple adjustments of the algorithm that would result in quasi-real-time behavior. In fact, it is a property of bottom-up parsing in general that quasi-real-time behavior is not guaranteed. Our parser has a less restrictive but similar property, ***fairness***—namely, our parser generates output that is linear in the input, though there is no constant over which output is guaranteed. For a fuller discussion of these properties, see Pereira and Shieber [97].

To summarize, preterminal delaying, as an intrinsic part of the algorithm, does not actually change the basic properties of the algorithm in any observable way. Note, however, that preterminal assignments, like reductions, are irrevocable once they have been made (as a by-product of the algorithm's determinism). Such decisions can therefore lead to garden paths, as they do for the sentences presented in Section 5.3.6.

We now discuss the central feature of the algorithm, namely, the resolution of shift-reduce conflicts.

5.3.3 The Disambiguation Rules

Conflicts arise in two ways: ***shift-reduce*** conflicts, in which the parser has the option of either shifting a word onto the stack or reducing a set of elements on the stack to a new element; ***reduce-reduce*** conflicts, in which reductions by several grammar rules are possible.

[4] I am indebted to Mitch Marcus for this observation and the previous comparison with his parser.

The parser uses two rules to resolve these conflicts:[5]

(1) Resolve shift-reduce conflicts by shifting.

(2) Resolve reduce-reduce conflicts by performing the longer reduction.

These two rules suffice to engender the appropriate behavior in the parser for cases of right association and minimal attachment. Though we demonstrate our system primarily with PP-attachment examples, we claim that the rules are generally valid for the phenomena being modeled [97].

5.3.4 Some Examples

Some examples demonstrate these principles. Consider the sentence

`Joe took the book that I bought for Susan.`

After a certain amount of parsing has been completed deterministically, the parser will be in the following configuration:

stack: NP V NP that NP V	*input:* for Susan	*state:* 23

with a shift-reduce conflict, since the V can be reduced to a VP/NP[6] or the P can be shifted. The principles presented would solve the conflict in favor of the shift, thereby leading to the following derivation:

stack: NP V NP that NP V P	*input:* Susan	*state:* 12
stack: NP V NP that NP V P NP	*input:*	*state:* 19
stack: NP V NP that NP V PP	*input:*	*state:* 24

[5]The original notion of using a shift-reduce parser and general scheduling principles to handle right association and minimal attachment, together with the following two rules, are due to Fernando Pereira [96]. The formalization of preterminal delaying and the extensions to the lexical-preference cases and garden-path behavior are due to the author.

[6]The "slash-category" analysis of long-distance dependencies used here is loosely based on the work of Gazdar [31]. The Appendix A grammar does not incorporate the full range of slashed rules, however, but merely a representative selection for illustrative purposes.

stack: NP V NP that NP VP/NP	*input:*	*state:* 22
stack: NP V NP that S/NP	*input:*	*state:* 16
stack: NP V NP $\bar{S}$	*input:*	*state:* 7
stack: NP V NP	*input:*	*state:* 14
stack: NP VP	*input:*	*state:* 6
stack: S	*input:*	*state:* 1

which yields the structure

$$[_S \text{Joe} [_{VP} \text{took} [_{NP} [_{NP} \text{the book}] [_{\bar{S}} \text{that I bought for Susan}]]]]$$

The sentence

Joe bought the book for Susan.

demonstrates resolution of a reduce-reduce conflict. At some point in the parse, the parser is in the following configuration:

stack: NP V NP PP	*input:*	*state:* 20

with a reduce-reduce conflict. Either a more complex NP or a VP can be built. The conflict is resolved in favor of the longer reduction, i.e., the VP reduction. The derivation continues:

stack: NP VP	*input:*	*state:* 6
stack: S	*input:*	*state:* 1

ending in an accepting state with the following generated structure:

$$[_S \text{Joe} [_{VP} \text{bought} [_{NP} \text{the book}] [_{PP} \text{for Susan}]]]$$

5.3.5 Lexical Preference

To handle the lexical-preference examples, we extend the second rule slightly. Preterminal-word pairs can be stipulated as either ***weak*** or ***strong***. The second rule becomes

(2) Resolve reduce-reduce conflicts by performing the longest reduction ***with the strongest leftmost stack element.***[7]

Therefore, if it is assumed that the lexicon encodes the information that the triadic form of "want" (V2 in the sample grammar) and the dyadic form of "position" (V1) are both weak, we can see the operation of the shift-reduce parser on the "dress on that rack" sentences of Section 5.2. Both sentences are similar in form and will thus have a similar configuration when the reduce-reduce conflict arises. For example, the first sentence will be in the following configuration:

stack: NP wanted NP PP	*input:*	*state:* 20

In this case, the longer reduction would require assignment of the preterminal category V2 to "want," which is the weak form; thus, the shorter reduction will be preferred, leading to the derivation

stack: NP wanted NP	*input:*	*state:* 14
stack: NP VP	*input:*	*state:* 6
stack: S	*input:*	*state:* 1

and the underlying structure

$$[_{S}\text{the woman}[_{VP}\text{wanted}[_{NP}[_{NP}\text{the dress}][_{PP}\text{on that rack}]]]]$$

In the case in which the verb is "positioned," however, the longer reduction does not yield the weak form of the verb; it will therefore be invoked, resulting in the structure

[7]Note that strength takes precedence over length.

$[_S\text{the woman}[_{VP}\text{positioned}[_{NP}\text{the dress}][_{PP}\text{on that rack}]]]$

5.3.6 Garden-Path Sentences

As a side effect of these conflict resolution rules, certain sentences in the language of the grammar will receive no parse by the parsing system just discussed. These sentences are apparently the ones classified as "garden-path" sentences, a class that humans also have great difficulty parsing. Marcus's conjecture that such difficulty stems from a hard failure of the normal sentence-processing mechanism is directly modeled by the parsing system presented here.

For instance, the sentence

```
The horse raced past the barn fell.
```

exhibits a reduce-reduce conflict before the last word. If the participial form of "raced" is weak, the finite verb form will be chosen; consequently, "raced past the barn" will be reduced to a VP rather than a participial phrase. The parser will fail shortly, since the correct choice of reduction was not made.

Similarly, the sentence

```
That scaly, deep-sea fish should be under-
water is important.
```

will fail, though grammatical. Before the word "should" is shifted, a reduce-reduce conflict arises in forming an NP from either "That scaly, deep-sea fish" or "scaly, deep-sea fish." The longer (incorrect) reduction will be performed and the parser will fail.

Other examples, e.g., "the boy got fat melted," or "the prime number few" would be handled similarly by the parser, though the sample grammar of Appendix A does not parse them [97].

5.4. Conclusion

To be useful, natural-language systems must model the behavior, if not the method, of the native speaker. We have demonstrated that a parser using simple general rules for disambiguating sentences can yield appropriate behavior for a large class of performance phenomena—right association, minimal attachment, lexical preference, and garden-path sentences—and that, morever, it can do so deterministically without generating all the parses and choosing among them. The parsing system has been implemented and has confirmed the feasibility of our approach to the modeling of these phenomena.

Appendix A. The Test Grammar

The following is the grammar used to test the parsing system descibed in the paper. Not a robust grammar of English by any means, it is presented only for the purpose of establishing that the preference rules yield the correct results.

S → NP VP	VP → AUX VP	$\bar{\text{S}}$/NP → that S/NP
S → $\bar{\text{S}}$ VP	VP → V0	S/NP → VP
NP → DET NOM	VP → V1 NP	S/NP → NP VP/NP
NP → NOM	VP → V2 NP PP	VP/NP → V1
NP → PNOUN	VP → V3 INF	VP/NP → V2 PP
NP → NP $\bar{\text{S}}$/NP	VP → V4 ADJ	VP/NP → V3 INF/NP
NP → NP PARTP	VP → V5 PP	VP/NP → AUX VP/NP
NP → NP PP	$\bar{\text{S}}$ → that S	INF/NP → to VP/NP
DET → NP 's	INF → to VP	
NOM → N	PP → P NP	
NOM → ADJ NOM	PARTP → VPART PP	

Appendix B. Sample Runs

```
>> Joe bought the book that I had been trying to obtain for Susan

Accepted: [s
```

```
            (np (pnoun Joe))
            (vp
              (v1 bought)
              (np
                (np (det the)
                    (nom (n book)))
                (sbar/np
                  (that that)
                  (s/np
                    (np (pnoun I))
                    (vp/np
                      (aux had)
                      (vp/np (aux been)
                             (vp/np (v3 trying)
                                    (inf/np (to to)
                                            (vp/np (v2 obtain)
                                                   (pp (p for)
                                                       (np (pnoun Susan]

>> Joe bought the book for Susan

Accepted: [s (np (pnoun Joe))
             (vp (v2 bought)
                 (np (det the)
                     (nom (n book)))
                 (pp (p for)
                     (np (pnoun Susan]

>> The woman wanted the dress on that rack

Accepted: [s (np (det The)
                 (nom (n woman)))
             (vp (v1 wanted)
                 (np (np (det the)
                         (nom (n dress)))
                     (pp (p on)
                         (np (det that)
                             (nom (n rack]

>> The woman positioned the dress on that rack

Accepted: [s (np (det The)
                 (nom (n woman)))
             (vp (v2 positioned)
                 (np (det the)
                     (nom (n dress)))
                    (pp (p on)
                        (np (det that)
                            (nom (n rack]

>> The horse raced past the barn fell

Parse failed.  Current configuration:
state:  (1)

stack:  <(0)>       [s (np (det The)
                           (nom (n horse)))
                       (vp (v5 raced)
                           (pp (p past)
                               (np (det the)
                                   (nom (n barn]


input:  (v0 fell)
        (end)

>> That scaly deep-sea fish should be underwater is important

Parse failed.  Current configuration:
state:  (1)

stack:  <(0)>       [s [np (det That)
                           (nom (adj scaly)
                                (nom (adj deep-sea)
                                     (nom (n fish]
                       (vp (aux should)
                           (vp (v4 be)
                               (adj underwater]


input:  (v4 is)
        (adj important)
        (end)
```

6. Generating Expert Answers through Goal Inference

This section was written by Martha Pollack.

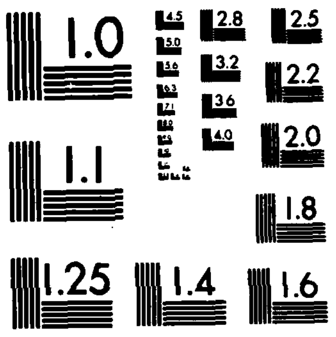
6.1. Introduction

Imagine that you're trying to use the mail system on the DEC-2060 computer, and that partway through creating a message you accidently type a Control-Z, which has the pernicious effect of ending your message creation and sending you to the prompt level. You turn to your ever-ready expert and ask if there's some way to delete a control character from within the mail system. Your expert tells you that there isn't, and so, feeling annoyed and frustrated, you proceed to recreate your message from scratch. When you later discover that you could indeed have entered an editor from the prompt level and added directly to the existing message fragment, you begin to doubt your expert's expertise.

Now imagine that you've just inherited $120,000. You immediately call an investment counselor to ask whether you'll earn more interest on Treasury-notes or on certificates of deposit. The counselor asks you a few questions about your tax bracket and other investments, and then informs you that T-notes are the better investment. What she doesn't tell you is that, given her knowledge of your financial situation, municipal bonds would be a far better investment than either of your expressed alternatives. When you later learn this fact, you're very likely to find a new investment counselor.

In each of these instances, you've gone to an expert seeking advice about some problem, and in each of them the expert has failed to provide you with the most appropriate advice. The failure resulted from the expert's assumption that you knew exactly what advice you needed, and that you had accurately and literally expressed a request for that advice in your query. That assumption leaves it up to you, the advice-seeker, like the classic traveler in Vermont, to know that you're in situation A and that you want to achieve situation B; the expert's role is simply to provide directions on "how to get there from here."

AD-A137 436   RESEARCH ON INTERACTIVE ACQUISITION AND USE OF   2/3
KNOWLEDGE(U) SRI INTERNATIONAL MENLO PARK CA ARTIFICIAL
INTELLIGENCE CENTER B J GROSZ ET AL. NOV 83
UNCLASSIFIED N00039-80-C-0575   F/G 9/2   NL


1.0
4.5
5.0
5.6
6.3
7.1
8.0
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

It is this restricted view of expertise that automated expert systems have adopted. Unfortunately, observation of dialogues between human experts and advice-seekers reveals that this picture bears little resemblance to reality. People often need to consult with experts precisely because they don't know what it is they need to know: they may have an incomplete or partial notion of what their options are—or perhaps no clear notion at all. As a result, they may intend doing something that is not actually the best thing they could do.

It is a significant feature of human expertise to be able to deduce, from an incomplete or inappropriate query, what advice is actually needed. The goal of this research is to develop a framework that will allow automated experts to perform similar deductions and thereby generate appropriate answers to queries made to them.

6.2. Overview and Related Research

The examples given in the introduction demonstrate that expert behavior necessarily involves the ability to determine an advice-seeker's unstated goals. This claim is an instance of the more general claim made by Pollack et al. [104] that,

> Expert systems, if they are to satisfy the legitimate needs of their users, must include dialogue capabilities as sophisticated as those proposed in current natural-language research. (p. 1)

Although it is almost a truism in natural language research today that recognizing a speaker's goals and intentions is prerequisite to understanding an utterance, the notion that goal detection is also a requirement for the successful provision of advice, e.g. by an expert system, is not as widespread.

Evidence that experts must often deduce what advice is necessary comes from the frequency with which they present indirect answers to the queries of those seeking advice. Hobbs and Robinson [47] distinguish among three sorts of indirect answers that can occur in discourse. The first, which they describe as indirect responses that nonetheless answer the question asked, is exemplified by responses to indirect speech acts. Thus, an expert answers "Do you know how I can print my file on the Imagen?" not with "Yes," but rather with the more helpful "Use the im command." Indirect speech acts, first discussed by Gordon and Lakoff [34] and Searle [115], are the subject of a growing body of linguistic literature. The beginnings of a formal account that would enable a natural-language component to deal with indirect, as well as direct, speech acts, is given in the work of Perrault, Allen, and Cohen, discussed below. Hirschberg [46] is working on a formal account of a particular class of indirect speech acts, which she terms scalar and hierarchical implicatures. These are typified by the question/answer pair: "Should I give the patient his medication?" "Give him the penicillin."

The second type of indirect answer discussed by Hobbs and Robinson is exemplified by responses that deny a pesupposition of the question asked. Kaplan [53] and Mays [84] both present methods for generating helpful answers to database queries that contain invalid presuppositions. Where those answers go beyond a simple statement of the invalid presupposition, their systems implicitly assume the goal of the question.[1] However, neither system is obviously extensible to a more general theory of question-answering.

Hobbs and Robinson describe their third category of indirect answers as consisting of those that address the higher goals of the question. In a sense this category subsumes the other two, since any cooperative response to a query cannot be random, but must address some goal of the speaker. Cohen, Perrault, and Allen (hereafter CPA) present an analysis [18, 99] of indirect speech acts that is explicitly tied to a process of inferring the plan, hence the "higher goal," of the speaker. Their general method for responding to an utterance by first inferring the speaker's goal and then cooperating in the achievement of that goal is similar to the method presented in this work. However, CPA focus only on the inferring of illocutionary goals, while in this work the inference of domain goals will be emphasized. To perceive the difference, consider the mail user who asked "Do you know how to delete a Control-Z?" CPA's theory would result in the hearer's inferring that this was an indirect request to be told the command that results in deletion of a Control-Z; the theory to be presented in this work would enable the hearer to infer that this indirect request is itself an indirect request for a method

[1]Kaplan suggests that the query "Is John a senior?" can be cooperatively answered by "No, a junior." This assertion assumes that the speaker's goal was to determine John's status. It would *not* be a cooperative response if the speaker's goal were instead to find a senior.

the user can employ to add increments to a partial mail message.

There are differences between the process of detecting and responding to an utterance's communicative goals and that of detecting and responding to its domain goals. When a speaker utters an indirect speech act, he is aware that it is indirect; futhermore, he expects his hearer also to recognize that and, if possible, to respond to the indirect question. In fact, he would consider rude a hearer who responded directly, e.g., by answering "Do you know how I can print my file on the Imagen?" with "Yes I do." Since the speaker knows that his hearer will have to infer his goal, he makes it a point not to make the required inference obscure. The fact that many indirect speech acts are conventionalized aids him in cooperative behavior and makes the hearer's task easier.

In contrast, when someone utters a request best answered with a response that addresses some domain goal not directly expressed by the query, he may well be unaware that this is so. Often he believes that what he's asking about is precisely what he needs to know. As we've already seen, it may not be; in that case, inferring what he really *does* need to know—by way of inferring what his domain goal actually is—may be quite complex. The speaker hasn't planned his request so as to facilitate the inference of goal: how could he have, when he was expecting a direct response?

While people all have roughly equivalent knowledge about communicative acts, there are gross imbalances in their knowledge about actions in any domain.[2] This imbalance is precisely what defines expertise. We shall bypass the question of deducing communicative goals—in fact, the illocutionary force of each utterance considered will be a request for information. Instead we shall concentrate on showing how an expert can exploit her greater knowledge of some domain to infer and respond to the domain goals of an utterance. However, it will be interesting to compare the mechanisms that enable each type of inference.

In another paper, Allen and Perrault [2] do incorporate this second level of plan inference

[2]Knowledge about communicative acts may not be strictly equivalent, since some people do seem to be better communicators than others, but the range of variation is probably much smaller: there aren't people who are experts at generating or understanding indirect speech acts, nor linguistically normal people who are unable to use them.

and, in fact, apply the same mechanisms in deducing both the communicative and domain goals of an utterance. The difference between their approach and ours is that they restrict themselves to answering questions in an extremely limited domain i.e, those that would be asked at a train station information booth, in which people are assumed to always have one of two simple goals. People attempting to achieve either of these goals are usually well aware of what they need to do so. If you want to meet an arriving train, you need only go to its arrival gate at its arrival time, and to do that you need to know its gate number and expected arrival time. Both the actions that attain the goal (of meeting the train) and the knowledge necessary to perform those actions are generally known to the agent who wants to achieve the goal, and so his questions to the expert will [accurately] ask for information he actually needs. Allen and Perrault deduce the higher domain goal of the speaker in order to furnish additional information that might be helpful to him: for example, if he asks for an expected arrival time, they want to provide a gate number as well. However, their assumption that the information the speaker requests is always what he truly needs to achieve his higher goal aids them in inferring that goal.[3]

In contrast, the domain of expertise in this work—MM mail system—contains a greater number of goals, which, significantly, can be structured in more complex ways. This by itself necessitates more complex inference techniques and control strategies than those presented by Allen and Perrault. Besides, in this domain people may not know what it is they need to *know* to achieve them. The MM user introduced in an example in Section 6.1 believed that, to finish creating his message, he had to delete the character that interrupted the creation process and, to accomplish this, he had to find out how to delete that character. In fact, he was mistaken

[3]Of course, this assumption may be incorrect even in the extremely simple domain considered by Allen and Perrault. Someone might ask an information booth attendant "What time does the Detroit train leave?" although, unbeknownst to him, there is no train to Detroit. If, however, there is a train to Chicago and a bus from Chicago to Detroit, the attendant should inform the advice-seeker of this fact. To do so, the attendant would have to be aware that the information the advice-seeker requested is not the information that will actually enable him to achieve his goal. Allen and Perrault's system would be unable to handle this case. Implicit in their heuristics for deciding among the possible goals of an advice-seeker is their assumption that the advice-seeker knows what information he needs. For instance, one of their heuristics decrements the probability that any plan containing an impossible referent is actually the advice-seeker's plan. In this example, the plan that involves the action "take the Detroit train" would be so decremented, since there is no referent for "the Detroit train." However this plan is precisely the one the expert needs to reason about to determine that the advice-seeker's goal is to get to Detroit.

about what actions would enable him to achieve his domain goal (of completing the message) and, consequently, also mistaken about what he needed to know.

Plan recognition for more complex plans, such as those the MM user might formulate, has been studied [15, 114, 25, 32]. However, all these works deal with ascertaining an agent's goal by observing his actions (or reading about them, as in Bruce's work). This is a completely different assumption from the one being made here, according to which the goal must be detected on the basis of a dialogue with the agent. The difference in assumptions becomes important when applications of this work are considered. For instance, if one wants to build an advice system that a computer user can invoke whenever he encounters a problem, it may well be infeasible to have that system constantly "watching over his shoulder." Moreover, if one wants to build an expert system that provides advice on some other domain outside the computer, it is likely to be impossible to have it watch over the advice-seeker's shoulder. Imagine an automated financial expert to whom a person could come to ask for advice on investments, taxes, real-estate purchases, and so forth—a computerized version of Dean Witter or Harry Gross, as it were.[4] Such an expert could receive its information only from discussion with the person seeking advice.

Appelt [4] presents a system for planning the generation of utterances and physical actions that is based on a theory of speech acts and communicative cooperation. In the examples he considers, the expert is always aware of the apprentice's goals and so does not have to do extensive goal inference. However, his methods of planning to help achieve that goal do, in many ways, parallel the methods presented here.

Also, both Carberry [16] and Litman [74] are concerned with tracking domain goals in discourse; Carberry has focused on the relationships among domain goals, Litman on the interaction between domain goals and communicative goals. Neither work, though, accounts for a speaker's requesting information that is *not* what he actually needs to achieve his goal.

Before developing the theory with which an expert can detect the possibly unexpressed

[4]Harry Gross is a financial expert who hosts a daily radio talk-show on WCAU in Philadelphia. Listeners call in with financial questions, and he advises them as best he can. Transcripts of his show provided the basis for an analysis of expert/advice-seeker interaction given by [104].

goals of a query and determine an appropriate answer, it is necessary set down an account of the process by which an expert determines the answer to a presumed direct query. To this end, a model of the advice-giving process will be given in the next section. In Section 6.4, this model will be applied to the *simple* case of expert question-answering, in which the advice-seeker does know what advice he needs, and moreover, makes a direct request for that advice. Then, in Section 6.5, we consider where and how this ideal behavior breaks down, and demonstrate a method for actively inferring the goal behind a question—rather than making the assumption that the question expresses it directly—and producing a response that addresses this goal. Finally, in Section 6.6, we outline the work that remains to be done.

6.3. A Framework for Describing Expert Answer-Giving

Before a method for generating indirect answers can be proposed, a framework must be adopted for describing the process by which an expert formulates and presents a direct answer. In a sense, deciding upon such a framework amounts to honing a definition of "expert system." The most obvious characteristics of expert systems are that they contain extensive information about some domain and that they can answer questions about that domain. While these are necessary characteristics, they are not sufficient in themselves. Simple database systems possess them, yet do not qualify as expert systems, since all they can do is retrieve specific facts requested by user. In some cases, database systems may also be able to perform simple computations upon those facts, but such computations must be specified by the user. For a system to be designated expert, it must have the capability to reason about its domain of expertise, that is, it must contain deduction rules as well as atomic facts.

Many "intelligent" computer system can be said to demonstrate expert behavior. While some have been called "expert system," other have instead been labeled "planning systems."

6.3.1 Expert Systems

Those programs that traditionally have been known as expert systems, e.g. MACSYMA [80], Prospector [39], Digitalis Advisor [35], have modeled expert reasoning with rule-based deduction systems, typically using heuristically controlled backwards chaining to deduce a conclusion. In these systems, knowledge about the domain of expertise is encoded in the system's axioms. The advice-seeker presents the system with the facts that constitute the problem description, often in response to questions asked by the system. What advice is to be given—what question the system is to answer—may be determined in one of three ways. The user may explicitly present the question he wants answered, as in the case of MACSYMA, where he specifically asks for, say, the solution to some differential equation. Alternatively, he may select the question he wants answered from a small set of questions proposed by the system. In Prospector, for instance, after asking for and receiving preliminary information about a particular site, the system presents several alternative hypotheses about what minerals might be present and then asks the user which hypothesis to pursue. Or, instead, the question to be answered may be constant, as in the Digitalis Advisor, where the answer is always how much much digitalis to give the patient.

6.3.2 Planning Systems

Other programs demonstrating expert behavior have been called planning programs, e.g. [24, 113]. These programs solve problems of how to convert one world state to another. The user of a planning system presents it with two descriptions: one of the initial state of the world, the other of the goal state he wishes to achieve. The solution derived by the system is a "plan" or sequence of actions to transform the problem state into the goal state. Special planning representations have been developed for these systems: STRIPS and the many systems modeled on it, for example, represent domain expertise in precondition- and add/delete-lists associated with each possible action. Plan synthesis techniques have also been developed: one notable extension of simple backwards chaining with precondition matching is the hierarchical expansion and criticism of NOAH.

6.3.3 Deduction or Planning?

A question that arises when attempting to construct a model of expert question-answering is whether the expert should do deduction or planning. The answer would seem to depend on the domain of expertise: certain domains involve reasoning about statements whose truth at some particular point in time is all that is of concern, while others involve reasoning about the effects of actions on the truth of certain statements. That is, some problems are more naturally modeled as deduction problems, others as planning problems. Problems presented to the Prospector system about the probability of some mineral's being present in an area according to facts known about that area, seem to be a case of the former, while those asked an MM expert about sequences of commands needed to perform some electronic mail task seem to exemplify the latter. It should be noted, though, that the very same expert may at certain times appear to be doing deduction and at other times planning: a financial expert could answer the question "Is there a limit on the amount I can deposit in an IRA?", which is most naturally described as a deduction problem, and also "How can I minimize my losses on a bad investment in silver that I made last year?", which is most naturally described as a planning problem.

Fortunately, a general model of expert problem-solving that does not force the deduction/planning distinction is possible. Rosenschein [111] and Kowalski [67] have shown separately that the distinction between deduction and planning systems is not intrinsic: planning systems can easily be embedded within inference systems. Kowalski demonstrates this by developing a planning system within the clausal logic of Prolog [131]: he employs a situation calculus that makes use of state variables to describe relations among states of the world. Rosenschein takes a different approach: he makes use of the dynamic logic that Pratt [105] originally developed to reason about program semantics. By using dynamic logic, Rosenschein is able to suppress state variables and instead exploit a dynamic necessity operator for capturing the effects of actions upon the world.

A brief description of Rosenschein's approach will demonstrate how planning is embedded in logic. The logic Rosenschein describes is a propositional logic that contains two atomic sets: the standard set of atomic propositions, and a set of atomic actions. The semantics of an atomic proposition is the standard mapping from that proposition to worlds, while the semantics of an atomic action is a mapping from that action to *pairs* of worlds. Intuitively viewed, an action α is mapped to the set of all pairs (w_1, w_2) such that if α is performed

```
     -----          -----
    |     |        |     |
    |  a  |        |  b  |
    |_____|        |_____|
    |     |        |     |
    |  b  |        |  a  |
    |_____|        |_____|
  -----------    -----------
     start          goal
```

Figure 6.3-1: A Trivial Planning Problem

in w_1, then w_2 will result. Compound propositions are built up in the standard way, using logical connectives, while compound actions are built up with connectives corresponding to "follows" and "nondeterministic choice." There are also syntactic rules for combining actions and propositions; the most important of these for our purposes is the rule that builds compound propositions, using the necessity operator "[ ]."[5] The compound proposition $[\alpha]p$ is true in any world w such that performing action α in w will result in p's holding.

Using this logic, Rosenschein is able to provide a formalization of planning. A planning problem consists of three parts: (1) a set of primitive (or atomic) propositions and actions; (2) a set of domain constraints representing the system's domain knowledge, where each constraint is either nonmodal (i.e., does not contain the necessity operator) or is of the form $p \supset [\alpha]q$, p and q being arbitrary nonmodal wffs; and (3) a set of plan constraints representing the problem to be solved, where these are also of the form $p \supset [X]q$. A solution to a planning problem therefore, is a [possibly compound] action β such that for each plan constraint, when X is replaced by β, the resulting sentence is provable from the domain constraints.

Although Rosenschein limited his discussion to the propositional case, Shieber [116] and Kautz [58] have independently worked out the details of using predicate dynamic logic for planning.

[5]Actually, in the formal syntax Rosenschein introduces the possibility operator "< >" to build a compound proposition from an action and a proposition. The necessity operator is later introduced as its semantic dual. However, both for Rosenschein's purposes and for mine, the necessity operator is more useful.

As an example of Rosenschein's approach, consider a trivial blocks-world problem shown in Figure 6.3-1. The start state for the example has two blocks, a and b, with a stacked atop b which is on a table; the goal state has b stacked atop a on the table. The primitive vocabulary[6] is:

Atomic Propositions:
$on(X, Y)$
$clear(X)$

Atomic Actions:
$transfer(X, Y, Z)$

The domain constraints express the usual blocks-world constraints, namely, that there are particular, named objects that are blocks; that there is another object, the table, which is not a block, which has a clear spot, and which is not on anything; that no block can both have something on it and be clear; and that you can transfer a block, provided it's clear, from one location to any other clear location.

Domain Constraints:

static:
$block(a)$
$block(b)$
$table(t)$
$\forall X \forall Y\ table(X) \supset \neg block(X) \wedge clear(X)\ and \neg on(X, Y)$
$\forall X \forall Y\ on(X, Y) \wedge block(Y) \supset \neg clear(Y)$

dynamic:
$\forall X \forall Y \forall Z\ on(X, Y) \wedge clear(X) \wedge block(X) \wedge clear(Z) \supset$
$[trans(X, Y, Z)] on(X, Z) \wedge clear(Y) \wedge clear(X)$

Finally, the single plan constraint is

plan constraint:
$on(a, b) \wedge on(b, t) \wedge clear(a) \supset [X] on(b, a) \wedge on(a, t) \wedge clear(b)$

A simple constructive proof, which will be left to the reader, leads to the not surprising instantiation of X as

$$trans(a, b, t); trans(b, t, a),$$

[6]Throughout this example, we will adopt the Prolog convention of beginning variables with an upper case letter and constants with a lower case letter.

where ";" denotes temporal ordering.[7]

Note that the solution the system would provide the user is the action β, i.e., the instantiation of the plan constraint's variable. This turns out to match Kowalski's approach closely, since Prolog, upon successfully proving the theorem that represents the planning problem, does precisely the same thing: it presents the user with the instantiations obtained in the successful proof.

An understanding of the relationship between planning and deduction permits a uniform treatment of expert systems, whether the problem being solved by them is most naturally cast as one of deduction or of planning. Both activities consist in performing logical deduction; however, while the theorem proved in the former case is static (nonmodal in Rosenschein's terminology), that proved in the latter may contain dynamic information.

An initial formulation of expert behavior, then, in the *ideal* case—i.e., when the advice-seeker knows and asks for what he needs—is as follows: expert advice-giving is a deductive process that uses axioms about the world. These axioms may include static or dynamic information. The advice-giving process is characterized by an advice-seeker who presents the expert with a set of facts relevant to some problem, which is itself cast as a theorem to be proved and which contains an uninstantiated variable. That theorem, which may or may not contain dynamic information, is either explictly presented by the advice-seeker or is constant and known to the expert. The latter attempts to prove the theorem from a combination of the axioms she knows and any facts applied by the user. If successful, she provides as a solution the instantiation she made of the variable in proving the theorem.

6.4. Providing Direct Responses

In the previous section, a model of the advice-giving process was presented. In this section, that model is applied to two simple examples. Both of these could be handled by

[7]I.e., $\alpha;\beta$ is defined to be true in all pairs of worlds (w_1, w_2) iff there is some w_3 such that α is true in pair (w_1, w_3) and β is true in pair (w_3, w_2).

existing expert or planning systems, since in each the advice-seeker does *directly* ask for the advice he actually needs. They are presented here to demonstrate the model just proposed.

6.4.1 The Domain

Throughout the rest of this section, the domain of expertise modeled is the TOPS-20 mail system, MM. Explanations of the way this system works will be provided with the examples as necessary. This domain is attractive because it permits several simplifications of the general problem of providing an appropriate answer:

- There is only one agent operating, apart from the system. Interactions among multiple agents' goals need not be considered.
- The effects of actions can be assumed to be certain. When a MM user types "SEND", we can assume that this results in his current message being sent. No such assumptions can be made about the effect, say, of investing money.
- There is a limited but nontrivial number of potential actions the MM user can perform. The number is small enough to make an axiomatization feasible. However, this small set of actions can be combined and structured in interesting ways.

As an example of the last point, notice that goals may be hierarchical. One possible goal might be described as "respond to Joe." A plan for achieving it could consist of finding and rereading the most recent message from Joe, sending a response, and moving Joe's message from your MM file into its own file in your directory. In turn, sending a response might consist of creating a response and sending it; creating a response might consist of typing part of message "on the fly" and inserting an already created file that contains the rest of the message. This plan is diagrammed in Figure 6.4-1.

The reader may have noticed that in the foregoing example the goal was described as an action. Technically, a goal is a set of formulas, not an action. However it is often easier to describe a goal in terms of an action that achieves it: in this example, for instance, it is easier

```
                        -------------------------
                        | respond to Joe's message|
                        |_________________________|
                                    |
                                    |
      ------------------------------|--------------------------
      |                             |                          |
      |                             |                          |
 ------v------              --------V--------          -----V--------
|find and read |           |    send a       |        | move message:|
|Joe's message |           |   response      |        | MOVE JOE.MAI N|
|______________|           |_________________|        |______________|
      |                             |
      |                             |
  ----|---------            --------|-------------
  |             |           |                     |
  |             |           |                     |
 _V_____    ____V___      __V______          _____V____
|HEADERS |  | TYPE N |   | ANSWER   |       |   SEND    |
|FROM JOE|  | where N|   |to enter  |       | to send   |
|to find |  |is the #|   | create   |       |the message|
| msg. # |  | found  |   | mode     |       |           |
|________|  |________|   |__________|       |___________|
                              |
                              |
                   -----------|------------
                   |                      |
                   |                      |
           --------V--------        ------V---------
          | enter beginning |      |   ESC  then    |
          | of message      |      |INSERT filename |
          |_________________|      |________________|

     Executable actions are capitalized.
```

Figure 6.4-1: Plan for "Respond to Joe's message"

to say "respond to Joe" than to list the set of propostions that responding to Joe achieves—i.e., have a message labeled "response" in Joe's MM file, have Joe's message out of your MM file, have Joe's message filed in its own file in your directory, and have nothing else changed. Hence, when we speak loosely of a goal as an action it should be interpreted to be the formula that that action achieves. Advice-seekers, for this reason *inter alia*, often describe their goal in terms of an action that they believe will achieve it; in Section 6.5.2 we consider how an expert can determine what someone's goal is—what formula he wants to hold—from a description of an action.

6.4.2 A Direct, Static Conclusion

Consider the following exchange between an advice-seeker, whose turn is labeled "A," and an expert whose turn is labeled "E":

A: "What is the abbreviation for TYPE?"

E: "TY"

This is an extremely simple case of advice-seeker/expert interaction. The advice-seeker knows exactly what it is he needs to know, and the expert can quite easily determine the necessary information for him. There are two alternative ways she might do so. The question is so simple that it might match one of her axioms directly, making its proof vacuous: it consists of a database lookup. In this example, the expert would have an explicitly encoded axiom stating that the abbreviation of "TYPE" is "TY":

$$\textit{\textbf{abbreviation}}(\text{``}TYPE\text{''}, \text{``}TY\text{''}).$$

Alternatively, the expert may know a rule that states that "Any command can be abbreviated to the shortest possible initial substring that is unique" and may have access to a list of all possible commands to determine that substring for "TYPE." In either case, the expert's task is to prove—using the axioms known to her—the following theorem:

$$\textit{\textbf{abbreviation}}(\text{``}TYPE\text{''}, X),$$

where X is an uninstantiated variable. Note that the instantiation of X is what counts as the solution.

It is interesting to see how our model of expert advice-giving handles a slight variation of this example, in which the advice-seeker asks his query in a yes/no form: "Is TY an abbreviation of TYPE?" This query, when translated in the obvious fashion into a theorem to be proved, doesn't contain an uninstantiated variable, but is rather of the form

$$\textit{\textbf{abbreviation}}(\text{``}TYPE\text{''}, \text{``}TY\text{''}).$$

There are two ways to fit this into our framework. The question is clearly about the truth-value of some proposition, so that truth-value can be regarded as the uninstantiated variable. Hence one solution is to imagine that the theorem to be proved is roughly of the form

$$\textit{truth-value}(\textit{abbreviation}(\text{"}TYPE\text{"}, \text{"}TY\text{"}), X)$$

and instantiate X as usual. Another, cleaner solution is to adopt the same approach as Prolog, and answer any theorem that doesn't contain an uninstantiated variable with a statement as to whether or not it is true.

6.4.3 A Direct, Dynamic Conclusion

Questions about how to achieve results seem to be naturally described in dynamic terms. Many of the questions that one might ask an MM expert are about commands to the MM system—e.g., "What does command C do?" or "What command should I use to achieve result P?" Since commands alter states, they are naturally treated as atomic actions and questions about them naturally regarded as planning problems. Consider this exchange:

A: "How do I get out of read mode?"

E: "If you've finished reading all the messages you said you wanted to read, a carriage return will get you out. If you want to stop early, just type QUIT."

Recall that the MM expert's knowledge includes a set of dynamic axioms describing the effects of commands on the state of the system. Among these axioms are the following:

(D1) $\textit{mode}(\textit{read}) \supset [QUIT]\textit{mode}(\textit{command})$
The QUIT command moves you from read mode to command mode.

(D2) $\textit{mode}(\textit{read}) \wedge \textit{all-msgs-read} \supset [< cr >]\textit{mode}(\textit{command})$
After all messages have been read, a carriage-return moves you from read mode to command mode.

(D3) $\forall x[\textit{mode}(x) \supset (\forall y\ \textit{mode}(y) \supset x = y)]$
Modes are mutually exclusive.

The theorem that the expert has to prove is

$$\textit{mode}(\textit{read}) \supset [X]\neg\textit{mode}(\textit{read})$$

The expert is able to prove this in two ways, and so presents both instantiations of X.

Let's first examine the proof that results in the instantiation of X to $QUIT$:

1.	$mode(read)$	assump
2.	$mode(read) \supset [QUIT]mode(command)$	D1
3.	$[QUIT]mode(command)$	mp,1,2
4.	$\forall x[mode(x) \supset (\forall y\ mode(y) \supset x = y)]$	D3
5.	$mode(command) \supset (\forall y\ mode(y) \supset command = y)$	∀-inst.,4
6.	$mode(command)$	assump
7.	$\forall y\ mode(y) \supset command = y$	mp,5,6
8.	$mode(read) \supset command = read$	∀-inst.,7
9.	$\neg command = read$	fact
10.	$\neg mode(read)$	mt,8,9
11.	$mode(command) \supset \neg mode(read)$	disch 6
12.	$[QUIT]mode(command) \supset \neg mode(read)$	necess.,11
13.	$[QUIT]\neg mode(read)$	dist. $\supset$,12,3
14.	$mode(read) \supset [QUIT]\neg mode(read)$	disch 1

The important lines in this proof are 1-3 and 14: the rest simple show that the command and read modes are mutually exclusive. As a result of this proof, the expert determines that $QUIT$ suffices as a solution to the advice-seeker's query. If this were the only proof of the problem theorem that the expert could construct, then her answer would probably have been simply "Use QUIT." However, she does have another way to solve the problem: she can attempt to use axiom D2 instead of D1 at line 2. In doing this, she discovers that she doesn't know whether or not the *all-msgs-read* condition holds. She then has two options. She can ask the advice-seeker—"Have you read all the messages you said to read?"—and then proceed according to his answer. Alternatively, she can adopt the approach of the sample dialogue and present her response as a conditional: "If you've finished reading all the messages you said to read, then [the plan I'm giving will work]." Deciding when to ask whether some condition holds and when to provide a conditional answer instead is a problem that goes beyond the scope of this work.[8]

[8]One other interesting aspect is the way in which the expert's determination that carrriage return is a possible solution can affect her statement that QUIT is a solution. In fact, QUIT is *always* a possible solution, but she says "If you want to stop early, use QUIT." This may be because she has some notion that QUIT is a more difficult solution than carriage return (four keystrokes more difficult). As a result, she conveys in her answer the information that there is an optimal solution—carriage return—but that it will apply only if certain conditions are met. When these conditions fail to hold—i.e., when the advice-seeker wants to stop *early*—then the nonoptimal solution of QUIT should be used.

6.5. Providing Indirect Answers

As was demonstrated by the examples in the introduction, an expert cannot assume that the most appropriate response to a query is a direct one. Even a query with a direct communicative goal may best be answered by a response that addresses some domain goal not directly expressed by the query. This section, discussing the reasons an advice-seeker might ask a question that requires such an answer, develops a method that the expert can use to generate one.

Section 6.5.1 explores two issues affecting expert answering that will not be central to the following. The first of these issues disappears by virtue of our assumptions about expert behavior. For the second, a sketch of a solution is proposed, although its details are left to future research. The two main ways in which the ideal picture of advice-giving already presented can fail to suffice comprise the topic of the next two sections: Section 6.5.2 discusses the situation in which the advice-seeker's query is something other than a statement of the goal formula; Section 6.5.3 discusses the situation in which the goal formula derived directly from his query is not the most appropriate formula for which to plan.

Of importance in this section are its explanations as to the nature of deviations from the ideal picture of advice-giving, and of the method the expert can apply to deal with such deviations, for the purpose of inferring the advice-seeker's domain goal and providing an appropriate answer. The specific inference rules, given by way of example, are preliminary: their details are still being developed.

6.5.1 Two Deferred Issues

6.5.1.1 Requests for Information vs. Requests for Action

One can conceive of a mail system expert capable of handling two sorts of requests: requests for information and requests for action. "How can I forward this message to Joe?" is a request for information, while "Can you forward this message to Joe?" is a request for action—at least when the latter is interpreted as an indirect question rather than a yes/no query regarding the expert's abilities. In the model of expertise being developed in this work, only requests for information will be handled.

The advice-seeker's goal, as inferred by the expert, describes the use to which he is likely to put the information he requests. When he asks "How can I forward this message to Joe?" the expert infers that his goal is roughly "this message is in Joe's mail file." If we were concerned with distinguishing between the two sorts of requests, this goal would be reserved for requests for action, while requests for information would be related to goals such as "knows how to achieve: this message is in Joe's mail file." However, since the expert behavior being modeled in this work is constrained to providing information (and not intervening actively on behalf of the advice-seeker), the "knows how to achieve" clause can be assumed to be implicit in the goal formula.

6.5.1.2 A Static Theorem as the Presented Goal

In the ideal picture of expert/advice-seeker interaction described in Section 6.3, the advice-seeker presents to the expert a theorem he wants proved. When the theorem is dynamic, it represents both the formulas that hold in the current state and those that should hold in the goal state; proving the theorem consists of constructing a plan that transforms the former into the latter. The static case has not received as much attention in this work to date, but it seems plausible that the expert transforms the static theorem into a dynamic one that captures the use to which an advice-seeker might put the information conveyed in the static theorem. For example, when the expert is asked "What is the abbreviation of TYPE?", she may reason that the advice-seeker wants to use that abbreviation; this deduction will enable her to determine the domain goal of the query. Allen and Perrault [2, p. 155] suggest several rules—the know-positive rule, the know-negative rule, and the know-value rule—that relate a request for information to a potential goal. A useful area for further research is the manner in which such rules as these apply in the model presented here.

6.5.2 Linking the Query to Possible Wants

Life would be simpler for the expert if advice-seekers always provided her with the formula they want to hold. Very often, though, the advice-seeker's utterance conveys something else. This section discusses what that something may be and how the expert can use it to deduce the goal.

6.5.2.1 Advice-Seeker Describes an Action

In Section 6.4.1 we saw that it is often easier to describe a goal in terms of an action that achieves it than in terms of the formula it comprises. This is partly a consequence of the frame problem: it is difficult to specify all and only the propositions whose truth will change as a result of some action, as was seen with the example of "respond to Joe." But there is at least one other reason goals are often most easily stated in terms of actions, namely, that a cluster of actions often acquires a single, commonly known name. "Respond to X" is one such cluster, where X is a parameter that may take the value "Joe," for instance. People have a general notion of what it means to respond to a message, and they expect the expert to share that knowledge. Note that the details of what it means to "respond to Joe" may differ from person to person—for instance, it may or may not include deleting Joe's message after the responding message is sent. The expert who is aware of these variations will either have to ask if the advice-seeker's notion of "responding to Joe" includes deleting Joe's message, or else provide alternative plans, one of which does include this and one that does not; there is simply no way for him to deduce which of them the advice-seeker wants, since the cluster of actions described by "respond to Joe" is indeterminate with respect to this.

If what the expert is going to provide is a plan that will achieve the advice-seeker's goal, then why does the advice-seeker describe to the expert a plan whose effect will be that goal? That is, if he already knows the action he wants to perform, why does he bother the expert at all? The answer to this is that the advice-seeker does not know *how* to perform the action he is describing: what he conveys to the expert is an action description, not an actual, executable plan. To keep this distinction straight, the term ***action description*** is used in this work to refer to the description given by the advice-seeker in his query, while the term *executable action* is used to refer to the action or cluster of executable actions that achieve corresponding results.

As an example of this distinction, recall the trivial blocks-world problem presented in Section 6.4. The problem was stated there as an example of the ideal case of expert advice-giving, and so the goal propositions were directly given. Imagine, however, that the question to the blocks-world expert had instead been "How can I stack block *b* on top of block *a*?" This question contains a description of an action that the advice-seeker wants performed: "stack X on Y," where X and Y are instantiated to *a* and *b* . "Stack" is not an executable action. The advice-seeker knows that "stack X on Y" is a description of an action (or cluster of actions) whose effect is commonly known; what he does not know is what sequence of executable actions will enable him to achieve that effect. The expert's task is to find an executable plan that achieves the results of his action description.

Sometimes the action description may be extremely close to an executable sequence of actions, particularly in a computer-system domain. This results in exchanges like the following:

A: "How do I delete a message?"

E: "Would you believe DELETE?"

Because the action description "delete a message" happens to be nearly identical to the MM command DELETE, it is easy to confuse the two. However, this dialogue is no different from the blocks-world one just described. In both cases, the advice-seeker gives a description of an action that is generally known; it just happens that MM has a single executable action whose name is nearly identical to the generally known action description that produces the same results. In another mail system, deleting a message might require that you first read the message and then type "DESTROY," so that the exchange between expert and advice-seeker instead looks like this:

A: "How do I delete a message?"

E: "Type READ n, where n is the number of the message you want to delete, and then type DESTROY."

Although the same action description is provided, the answer consists of a different executable plan.

Recall that, in the initial formulation of advice-giving, the advice-seeker directly informs the expert of the goal formula. When the advice-seeker presents something else in his query, such as an action description, the expert needs rules to link the query to a goal. The first of these rules, which will be called *linking rules*, applies when the advice-seeker presents an action description:

Linking Rule 1 (LR1).

$$WantAction(A, \alpha) \wedge Believes(A, achieves(\alpha, p, w_0)) \supset POS\ want(A, p)$$

This rule states that if some advice-seeker A describes an action α that he wants performed, and if he believes that some arbitrary wff p is the result of performing that action in the current world, then p may be his goal.[9]

There are several things to note regarding this rule. First, the condition is stated in terms of the advice-seeker's beliefs. Remember that action descriptions, while commonly known, may not be fully determined with respect to their results. For instance, certain people believe that performing the action "respond to Joe" entails the result "Joe's last message is deleted," while others do not believe this. Whether or not this proposition is part of the advice-seeker's goal depends upon his beliefs. The *Believes* relation, in fact, is where the common knowledge about action descriptions is stored. The examples in the next section will demonstrate how it is used.

Secondly, the goal formula has the modal possibility operator in front of it. This is because the most the expert can ever say is that an advice-seeker *might* want some goal. As rules for goal inference are introduced, the set of formulas the expert can infer as possibly deserved by the advice-seeker will grow. In certain cases, she may even infer that he might want both p and $\neg p$. This is not necessarily because the advice-seeker has inconsistent wants,

[9]This entire linking rule is within the context of the expert's belief, i.e., a more precise statement of it would be $Believes[E, (WantAction(A, \alpha) \wedge Believes(A, achieves(\alpha, p, w_0)))] \supset Believes(E, POS\ Want(A, p))$. Since all the deduction described in this work is within the context of the expert's belief, the outer predicate will be assumed implicit to keep the formulas simpler.

but because the expert lacks enough information to determine which he actually wants. The possibility operator provides a way to model this situation correctly.

Also, the logic used is multisorted: variables may range over plans, formulas, or possible worlds. To keep things straight, variables ranging over plans will be symbolized by Greek letters, those over logical formulas will be (possibly subscripted) p's and q's, and those over possible worlds will be (possibly subscripted) w's. The current world is denoted by w_0. When x, y, and z appear as variables, they range over individual terms within the formulas denoted by the p's and q's.

Although the second sort ranges over logical formulas, the use of a second-order logic can be avoided by employing Moore's trick of doing deductions indirectly by using the first-order formalization of the semantics (see [90] for details). This is important because automatic methods for doing second-order deduction are not well understood. All operators whose arguments can be formulas must be treated as abbreviations for their semantic expansion. The expansion for *POS* and *want* is as follows:

$$POS(p) \equiv \exists w T(w,p)$$
$$T(w, want(A,p)) \equiv \forall w_1 [\omega_A(w,w_1) \supset T(w_1,p)],$$

where ω_A is the want accessibility relationship for agent A, and $T(w,p)$ holds whenever p is true in world w. (See [4, pp. 56–62] for more details of a possible worlds semantics for wanting.) By joining these two definitions, we derive

$$POS\ want(A,p) \equiv \exists w \forall w_1 [\omega_A(w,w_1) \supset T(w_1,p)]$$

The expansion of *achieves* is

$$achieves(\alpha,p,w) \equiv \exists w_1 [R(\alpha,w,w_1) \land T(w_1,p)] \land \forall w_1 [R(\alpha,w,w_1) \supset T(w_1,p)].$$

R is the result relation: $R(\alpha,w_1,w_2)$ holds if and only if performing α in w_1 can result in w_2. The reason both halves of the conclusion are necessary is that, without the existential clause, an action might be said to achieve a result when in fact that action never terminates; without the universal clause, an action might be said to achieve a result when in fact that result is accidental and only happens sometimes when the action is performed.

The *achieves* relation is intended to capture the important effects of an action. After an action has been performed, two types of propositions hold: (1) those that have been made true by the performance of the action; (2) frame propositions that were true before the action was performed. (*Frame propositions* are propositions are propositions whose truth-value is not affected by the performance of the action.) In most analyses of planning and actions, all frame propositions have equal status. However, when someone wants to perform an action, he will be more concerned with some frame propositions than with others. The *achieves* relation explicitly mentions not only those propositions that an action changes, but also those that critically it does not change.[10] If $achieves(\alpha, p, w)$ holds, then p is the conjunction of propositions whose value changes as a result of α and the critical set of propositions whose values do not change.

Consider, for instance, the MM user who has finished his message and now wants to move from create mode to send mode. Not only does he want the mode to change, but it is essential to him that his message remain unchanged; after all, the reason he is shifting into send mode is to send the message he's just created. On the other hand, it probably does not matter to him whether his terminal display remains similarly unaffected. A plan that enables him to get to send mode but changes his current message will be unsatisfactory to him, whereas one that gets him to send mode but clears the display will suffice.

To use LR1 (the linking rule just proposed) a restatement of the expert's top-level task is needed. Although the earlier formulation assumed that she was given the goal formula, actually it is part of her job to deduce it. Only after that deduction can she produce a plan to achieve the goal; as before this plan will serve as her answer. Expressed formally, the advice-giving task can be described as follows:

Expert's Top-Level Task:

From the domain axioms and the linking rules, prove

[10]This is oversimplified for the time being: which propositions critically cannot change may depend on why the action is being performed. That is, at some times one proposition may be a critical frame proposition, and at other times it may not be. This problem will need to be addressed in future work.

$$F \wedge T \supset \exists\alpha\exists p POS\ want(A,p) \wedge achieves(\alpha,p,w_0)$$

where $F = \{\, facts\ about\ the\ current\ world, w_0 \,\}$

$T = \{\, translation\ of\ the\ query \,\}$

and $A = the\ advice\ seeker$.

In constructing the proof, the expert instantiates two variables: p, representing a possible goal of the advice-seeker, and α, a plan that achieves that goal. It is the instantiation of the latter that will become the expert's answer. The possible wants are deduced from the translation of the query by using the linking rules, and the plans are deduced by using the expert's domain knowledge.

We do not consider here how the query would be translated from English. Instead only a few target translations are considered. An examination of transcripts of expert/advice-seeker dialogues has shown that many queries can be paraphrased as "How can I have condition P hold?" (where P may be some condition or a Boolean combination of conditions), "How can I do action α?" (where α may be some action or a combination of actions), or "How can I do action α but have condition P hold?"[11] These correspond, respectively, to the predicates $WantFormula(A,p)$, $WantAction(A,\alpha)$, and $WantModifiedAction(A,\alpha,p)$. The idea is to leave the requisite semantic analysis unspecified: any mechanism for semantic analysis could provide logical forms that could then be translated in a principled way into such a set of predicates.

[11]The transcripts examined came from four sources: (1) a session between a user who was trying to learn MM and an expert; (2) a session between a user who was trying to learn EMACS and an expert; (3) several sessions between naive users who were trying to accomplish a task in EMACS and an expert who was advising them; (4) dialogues between Harry Gross, the financial expert mentioned previously, and radio listeners who called him for advice. In addition to the three types of questions mentioned above, there is a fourth category of commonly asked questions: these occur when something has "gone wrong," and the advice-seeker asks either "How did I end up with condition X holding (when I wanted condition Y)?" or "I performed action A, what conditions hold now?" Analysis of this last class has been deferred because it seems to require an additional layer of reasoning. In the question types considered in this work, there are two states of interest to the expert: the current state and the goal state. For queries falling into the fourth type, there is a third state of interest: the one that held before the advice-seeker attempted to achieve his goal. The expert may need to deduce what that state was or what the current state is (or both). She also must decide whether to tell the advice-seeker how to achieve his goal directly from the current state, or how to return first to the previous state and achieve his goal from there.

6.5.2.2 Some Examples

We are ready to present some examples of expert advice-giving. Two such examples will be given in this section. First, because it is simple, we shall repeat the example of Section 6.4.3, in which the advice-seeker present a formula he wants to hold. We shall then follow with an example in which the advice-seeker furnishes an action description.

The only linking rule provided so far relates a given action description to a possible goal. The rule linking a given formula to a possible goal is even simpler:

Linking Rule 2 (LR2).

$$WantFormula(A, p) \supset POS\ want(A, p)$$

In other words, if an advice-seeker gives a formula that he wants to hold, that formula may be his domain goal. (In Section 6.5.3 we shall see that it is not necessarily the correct domain goal to which to address a response.) With this second linking rule, the expert can respond to "How do I get out of read mode?"

A: "How do I get out of read mode?"

E: "If you finish reading all the messages you said to read, a carriage return will get you out. If you want to stop early, just type QUIT."

Translation
(T1) $WantFormula(A, \neg mode(read))$

Facts about w_0
(F1) $T(w_0, mode(read))$

Linking Rules
(LR2) $\forall p[WantFormula(A, p) \supset POS\ want(A, p)]$

Domain Axioms
(D1) $\forall w[T(w, mode(read)) \supset achieves(QUIT, mode(command), w)]$

(D2) $\forall w[T(w, mode(read) \wedge all\text{-}msgs\text{-}read) \supset achieves(< CR >, mode(command), w)]$

(D3) $\forall w[T(w, mode(command)) \supset T(w, \neg mode(read))]$

D1 and D2 are direct translations of axioms D1 and D2 from the proof given in Section

6.4.3. D3 is actually a lemma. Since a proof that it follows from a more general statement of the exclusiveness of modes was a subproof in Section 6.4.3, we'll use it as a lemma here so as to shorten the proofs. In any case, it is secondary to the major concerns of this section.

Actually, in a complete planning system the expert would need to know the entire set of frame properties, not just the critical ones expressed by *achieves*. To represent this, another predicate, ***result***, can be introduced. ***result*** is similar to *achieves*, except that it includes a way of expressing the frame property. It is a four-place predicate, defined as follows:

$$\begin{aligned}result(\alpha,p,f,w) \equiv & \exists w_1[R(\alpha,w,w_1) \wedge T(w_1,p) \wedge [\forall q(f(q) \supset [T(w_1,q) \equiv T(w,q)])]] \wedge \\ & \forall w_1[R(\alpha,w,w_1) \supset [T(w_1,p) \wedge [\forall q(f(q) \supset [T(w_1,q) \equiv T(w,q)])]]].\end{aligned}$$

This relation says that, if α is performed in world w, it is guaranteed to terminate and that, in the resulting world w_1, p will hold; furthermore, any proposition q that has the frame property f will hold according to whether it held in w. As with *achieves*, p expresses both those propositions that change and those that critically stay constant. The frame property f is given formally as a λ-expression: a proposition r is a frame proposition (i.e., its truth-vaue is unaffected by the action α) if a true expression results when r is substituted for the bound variable in f.

The complete specification of the actions QUIT and <CR> is then

(D1') $\forall w[T(w, mode(read)) \supset$
$result(QUIT, mode(command), \lambda p[\neg mode\text{-}incompat(command, p)], w)]$

(D2') $\forall w[T(w, mode(read) \wedge all\text{-}msgs\text{-}read) \supset$
$result(<CR>, mode(command), \lambda p[\neg mode\text{-}incompat(command, p)], w)]$

In addition, the following axioms specify the frame exceptions:

(D4) $mode\text{-}incompat(command, all\text{-}msgs\text{-}read)$

(D5) $\forall x \forall y[\neg(x = y) \supset mode\text{-}incompat(y, mode(x))]$

mode-incompat is a two-place relation whose argument is a mode name and whose second argument is a formula that is incompatible with the named mode. D4, for instance, says that the proposition *all-msgs-read* does not make sense in command mode; D5 says that two distinct instantiations of the mode relation cannot hold at once. The *mode-incompat* relation is used in the frame clauses of D1' and D2' so that all propositions except those satisfying *mode-incompat(command, x)* will continue to hold when the mode changes from read to command.

Although the *result* relation would be needed in a complete planning system, *achieves* will be sufficient for the examples given here, especially in light of the following lemma relating the two, which follows directly from their expansions:

(R1) $\forall\alpha\forall p\forall w[result(\alpha,p,f,w) \supset achieves(\alpha,p,w)]$.

A second lemma that will help shorten the proofs is

(R2) $\forall\alpha\forall p\forall w[[achieves(\alpha,p,w) \wedge [T(w,p) \supset T(w,q)]] \supset achieves(\alpha,q,w)]$.

This also follows directly from the expansion of *achieves*.

The constuctive proof of the top-level theorem, which corresponds to answering "How do I get out of read mode?" follows. Notice that, in lines 1 and 2, T1 and LR1 are introduced as assumptions that are then discharged in line 11 to prove the top-level implication.

1.	$WantFormula(A, \neg mode(read))$	T1
2.	$T(w_0, mode(read))$	F1
3.	$\forall p[WantFormula(A,p) \supset POS\ want(A,p)]$	LR2
4.	$POS\ want(A, \neg mode(read))$	mp,1,3
5.	$\forall w[T(w, mode(read)) \supset achieves(QUIT, mode(command), w)]$	D1
6.	$T(w_0, mode(read)) \supset achieves(QUIT, mode(command), w_0)$	∀-inst.,5
7.	$achieves(QUIT, mode(command), w_0)$	mp,2,6
8.	$\forall w[T(w, mode(command)) \supset T(w, \neg mode(read))]$	D3
9.	$achieves(QUIT, \neg mode(read), w_0)$	R2,7,8
10.	$POS\ want(A, \neg mode(read)) \wedge achieves(QUIT, \neg mode(read), w_0)$	4,9
11.	$WantFormula(A, \neg mode(read)) \wedge T(w_0, mode(read)) \supset$ $POS\ want(A, \neg mode(read)) \wedge achieves(QUIT, \neg mode(read), w_0)$	disch,1,2

Hence the expert is able to find a plan, namely QUIT, that achieves a possible goal, namely $\neg mode(read)$. As before, she could instead have chosen here to use rule D2 in line 5, and would then have faced the decision of how to deal with not knowing whether *all-msgs-read* holds in w_0.

Now let's examine a more interesting case, in which the expert has to determine a possible goal from an action description:

A: "How do I delete a Control-Z? I hit it by accident and now I'm in send mode."

Translation
(T1) $WantAction(A, delete(control\text{-}z))$

Facts about w_0
(F1) $T(w_0, mode(send))$
(F2) $T(w_0, msg(x_0))$

Linking Rules
(LR1) $WantAction(A, \alpha) \wedge Believes(A, achieves(\alpha, p, w_0)) \supset POS\ want(A, p)$

Domain Axioms
(D1) $\forall w \forall x[T(w, mode(create) \wedge msg(x)) \supset achieves(control\text{-}z, mode(send) \wedge msg(x), w)]$

(D2) $\forall w \forall x \forall y[T(w, mode(create) \wedge msg(x)) \supset achieves(insert(y), mode(create) \wedge msg(x \circ y), w)]$
where "∘" denotes concatenation

(D3) $\forall w \forall x \forall y[T(w, mode(edit) \wedge text(x)) \supset achieves(insert(y), mode(edit) \wedge text(x \circ y), w)]$

(D4) $\forall w \forall x[T(w, mode(edit) \wedge text(x)) \supset achieves(control\text{-}x; control\text{-}z, mode(send) \wedge msg(x), w)]$

(D5) $\forall w \forall x[T(w, mode(send) \wedge msg(x)) \supset achieves(EDIT, mode(edit) \wedge text(x), w)]$

What does the expert know the advice-seeker believes about the action description "delete?" In general, the problem of determining what clusters of actions have commonly known descriptions is a difficult one to which further work should be devoted. The present work offers only a preliminary solution to dealing with the "delete" description in this query.

One action description that is probably well formed defines the parameterized action description *undo*(α) in terms of the action description α:

$$\exists p \exists q[Believes(A, \forall w_1[T(w_1, p) \supset achieves(\alpha, q, w_1)]) \supset$$
$$Believes(A, \forall w_2[T(w_2, q) \supset achieves(undo(\alpha), p, w_2)])]$$

That is, if the advice-seeker believes that α achieves q from a world in which p holds, then he believes that undoing α from a world in which q holds should result in p.

For the current example, we shall assume that the advice-seeker is equating "delete" and "undo": in asking to "delete a Control-Z," he is asking to "undo a Control-Z" whose effect was to send him from create mode to send mode. There is an unfortunately large logical leap required in making this assumption. "Delete" is actually closer to the inverse of "insert": the action described by "deleting" is the action of removing a character that has been inserted effectively undoing the insert. Control characters are not characters that can be inserted, though, so, if the expert stopped at this point, her response would be "You can't delete control characters" because you can't even insert them. Instead a competent expert determines that what the advice-seeker really wants is to undo the effect of typing Control-Z. Perhaps she arrives at this conclusion by analogy: just as deleting the last noncontrol character restores the state of the system to the one that immediately preceded the insertion, deleting a control character may be thought of as restoring the system to the state it had been in before the control character was typed.

Precisely how an expert can use methods such as analogy to deduce what it is an advice-seeker believes about an action description is a question that must be addressed in the future. This deduction is just the first step in the expert's inference process: after applyinng methods such as analogy to ascertain the advice-seeker's beliefs about the action he's described, the expert must next apply the linking rules to determine a possible goal. After this she may apply alternative-goal rules, which will be introduced in Section 6.5.3, to expand the set of possible goals. The following will focus on the linking rules and the alternative-goal rules; for now we shall assume that, whether by anology or by some other means, the expert is able to relate an action description to the set of propositions that A believes are its effects. In this example she determines that A, in asking to "delete a Control-Z," is actually asking to "undo a Control-Z."

With this assumption the expert can deduce

$$Believes(A, \forall w \forall x[T(w, mode(send) \land msg(x)) \supset achieves(delete(control\text{-}z), mode(create) \land msg(x), w)])$$

from the instantiation of the "undo" action description with

$$Believes(A, \forall w \forall x[T(w, mode(create) \land msg(x)) \supset achieves(control\text{-}z, mode(send) \land msg(x), w)]).$$

She can then apply linking rule LR1 and the fact that the advice-seeker knows that $mode(send)$ holds in w_0 to deduce that a possible goal of the advice-seeker is $[mode(create) \land msg(x_0)]$. Unfortunately, when she then attempts to complete the top-level proof and construct a plan to achieve $mode(create)$, she fails. There simply isn't any way in MM to return to create mode from send mode. At this point, she needs to find some other possible goal of the advice-seeker. Section 6.5.3 will discuss how she can do that.

6.5.2.3 Advice-Seeker Gives Something More Complex

In all the examples discussed so far, the advice-seeker gave either a single proposition or a single action description. In fact, he may present arbitrary Boolean combinations of actions or propositions. To handle these, the expert needs additional linking rules. Their exact formulation is largely a matter for future research, but a few such rules are discussed here to show what issues are involved therein.

6.5.2.3.1. Disjunction of Action Descriptions

In one common form of query, advice-seekers present a disjunction of action descriptions. An example of this appeared in our introduction, in which we described an advice-seeker who wanted to invest in either Treasury-notes or certificates of deposit. What a competent expert would have deduced from his request is that he wants to make some investment that will provide him interest. This deduction arises because investing in T-notes and investing in CDs both customarily achieve that result.[12] In general, whenever the expert is presented with alternative action descriptions, a possible goal will consist of the intersection of the sets of effects the advice-seeker believes those actions will achieve. The appropriate linking rule will look something like the following:

Linking Rule 3 (LR3).

$$WantAction(A, OR(\alpha, \beta)) \wedge Believes(A, achieves(\alpha, p_1, w_0)) \wedge Believes(A, achieves(\beta, p_2, w_0)) \supset POS\ want(A, P)$$

where P represents the conjunction of all those propositions belonging both to p_1 and p_2.

6.5.2.3.2. Conjunction of Propositions

A second common form of query involves giving a conjunction of propositions that the advice-seeker wants to hold. Linking rule 2, as it is now stated, will apply in this case, so that if an advice-seeker says he wants $(p \wedge q)$ to hold, the expert will deduce that $(p \wedge q)$ is a possible

[12]In fact, his goal is more than just to receive interest: what he wants is to receive maximum interest. The expert is able to deduce this because of domain knowledge not accounted for here. In fact, such domain knowledge is necessary even to select the goal from among the common consequences of investing in T-notes and investing in CDs.

goal. A difficulty arises, however, since, in any system that reasons about wants, it is natural for the following axiom to exist:

$$\forall p \forall q\, Want(A, p \wedge q) \supset Want(A, p) \wedge Want(A, q)$$

Starting with $WantFormula(A, p \wedge q)$, the expert, in constructing her top-level proof, might apply this rule. As a consequence she would deduce that p is a possible goal of the advice-seeker, and then might attempt to find a plan to achieve just p. If she succeeds at this and finds a plan that achieves p but not q, the current formulation of the expert's task would regard this plan as an appropriate answer, since it does achieve a possible want. In fact, it is inappropriate, since the advice-seeker wants $(p \wedge q)$, not just p.

A possible solution to this problem is again to reformulate the statement of the expert's top-level task.[13] Instead of inferring *any* possible goal of the advice-seeker, the expert will be required to find a possible goal that is "maximal" with respect to the set of possible goals. A partial ordering of goals is defined by logical implication, so that $p > q$ if and only if $p \supset q$. Then a possible goal p is said to be maximal if and only if there is no other possible goal q, such that $q \supset p$. Hence if $(p_1 \wedge q_1)$ is a possible goal, p_1 will also be a possible goal, but it will not be maximal, since $(p_1 \wedge q_1) \supset p_1$. This restriction of acceptable solutions to those that satisfy maximal goals also precludes the problem of attempting to plan for extra disjunctions: if p_1 is known to be a possible goal, then, for arbitrary q_1, $(p_1 \vee q_1)$ will also be a possible goal. However, $(p_1 \vee q_1)$ will not be maximal, and so the expert will not attempt to plan for it.

Formally, the restatement of the expert's top-level task is

Expert's Top-Level Task:

From the domain axioms, action descriptions, and linking rules, prove

$$F \wedge T \supset \exists \alpha \exists p M(p) \wedge achieves(\alpha, p, w_0),$$

where $F = \{$ facts about the current world, $w_0 \}$

and $T = \{$ translation of the query $\}$.

[13]This solution was suggested by Stan Rosenschein.

To expand $M(p)$, we need to define max, the standard predicate expressing maximal within a partial order:

$$\forall x \forall set \forall ord[max(x, set, ord) \equiv in(x, set) \wedge \forall y[in(y, set) \supset \neg greater(y, x, ord)]]$$

When ord is instantiated for implication, the definition of $greater$ is

$$\forall x \forall y[greater(x, y, impl) \equiv \forall w[T(w, x) \supset T(w, y)]]$$

Finally, a formula p is in the set $posswants$ of agent A if and only if it is a possible want:

$$\begin{aligned} &\forall p[in(p, posswants_A) \equiv \exists w \forall w_1[\omega_A(w, w_1) \supset T(w_1, p)]] \\ i.e., \; &\forall p[in(p, posswants_A) \equiv POS\ want(A, p)]. \end{aligned}$$

$M(p)$ now expands as follows:

$$\begin{aligned} M(p) &\equiv max(p, posswants_A, impl) \\ &\equiv in(p, posswants_A) \wedge \forall q[in(q, posswants_A) \supset \neg greater(q, p, impl)] \\ &\equiv POS\ want(A, p) \wedge \forall q[POS\ want(A, q) \supset \neg \forall w[T(w, q) \supset T(w, p)]] \\ &\equiv POS\ want(A, p) \wedge \forall q[POS\ want(A, q) \supset \exists w[T(w, q) \wedge \neg T(w, p)]] \end{aligned}$$

So, for example, if $(p_1 \wedge q_1)$ and p_1 are both possible wants, p_1 will fail to satisfy M, since there is no world in which $(p_1 \wedge q_1)$ is true but p_1 is false.

Using this redefinition of the top-level goal is tricky, however, since it is impossible to prove that a formula is completely maximal. Instead, the problem must be constrained to specify that a formula is maximal with respect to the formulas that are *provably* possible wants. The constraint process is similar to McCarthy's circumscription process [85], and can probably be achieved by making all the linking rules biconditional. However, because this approach does not seem cleanly to capture intutive notions of the goal inference process, alternative solutions are being considered.

6.5.2.3.3. Action Description and Modifying Proposition

One other common form of query presents both an action description and a modifying proposition (or possibly a Boolean combination of modifying propositions). This is exemplified by the following interchange:

A: "How can I move a mail message to another file but not have it deleted from the MM file?"

E: "Use COPY instead of MOVE."

These queries occur when an advice-seeker knows a common action description whose associated effects are approximately what he wants. He presents that description and either the associated effects he does not want or else some additional effects he does want.

Such requests can be translated to the target predicate *WantModifiedAction*, which gives both the desired action description and the modifying propositions. The linking rule to handle this predicate is:

Linking Rule 4 (LR4).

$$WantModifiedAction(A,\alpha,p) \wedge Believes(A, achieves(\alpha, \neg p \wedge q, w_0)) \supset POS\ want(A, p \wedge q).$$

6.5.2.3.4. Summary

In all, the five major types of queries to an expert that have been identified so far are those that (1) describe an action, (2) describe a disjunction of actions, (3) present a goal proposition, (4) present a conjunction of goal propositions, and (5) describe an action but also give modifying formulas. There are almost certainly other types of queries that advice-seekers can direct to experts; the identification of such additional types will be an objective of future research.

6.5.3 Linking Possible Goals to Other Goals

Recall the plight of the expert who, in the last section, was attempting to answer the query "How can I delete a Control-Z?" Although she was able to deduce that what the advice-seeker wanted was to return to create mode, she was unable to provide him with a plan to achieve that goal. Her knowledge of the domain led her to answer him with merely "You can't get there from here." The human expert who actually responded to this query, however, was able to provide a much more appropriate answer. She told the advice-seeker "Type EDIT to enter the editor. Then you can finish building your message and, when you're done, type Control-X Control-Z to return to send mode." To arrrive at this answer, the human expert had to deduce that the reason the advice-seeker wanted to be in create mode was to finish building his message prior to sending it. He believed that being in create mode was prerequisite to being in send mode with a completed message.

An expert often needs to make just this sort of deduction. It is not always sufficient for her to determine the advice-seeker's possible goals directly from his query: often she will have to determine why those goals are possible goals, and what other goals they are intended to support. The introduction of a set of rules called *alternative goal rules* will enable this type of reasoning in the framework being proposed. Alternative goal rules add to the set of possible goals: they are all of the form "if p is a possible goal, then q is also a possible goal."

Allen and Perrault [2] present a set of rules that interconnect an advice-seeker's possible wants. For example, they propose a rule stating that, if an advice-seeker wants some proposition to hold and if that proposition is a precondition of some action, then the advice-seeker may want to perform that action (Precondition-Action Rule). Another rule states that, if an advice-seeker wants to perform some action, then he may want the effect of that action (Action-Effect Rule).

A major thrust of the continuation of this work will be to identify and formalize alternative-goal rules such as these. This section presents just one such rule, corresponding to a combination of Precondition-Action rule and the Action-Effect rule, and shows how its use results in the expert's answering the question "How can I delete a Control-Z?" appropriately.

The alternative goal rule needed by the expert is

Alternative Goal Rule 1 (AG1).

$$\forall p \forall q \forall \alpha[[POS\ want(p) \wedge \forall w[T(w,p) \supset achieves(\alpha, q, w)]] \supset POS\ want(A, q)].$$

This rule asserts that, if p is a possible goal of the advice-seeker and p is a precondition of some action α, then another possible goal of the advice-seeker is what α achieves, namely q. Notice that the implication in the second conjunct of the hypothesis goes in one direction only: p need not be a necessary precondition of α for this rule to apply. Suppose there are several alternative sets of preconditions of α; as long as one of them holds, α can be performed. In this case, the advice-seeker may want any one of the sets of preconditions to hold if what he wants to do is perform α. Notice too that p need not be sufficient precondition of α for AG1 to apply. There may be other preconditions in the set containing p that the advice-seeker, since he already knows how to achieve them, does not ask about. These other preconditions are represented by variable r in AG1. We allow r to be instantiated by the logical constant "true" in the case that p is a sufficient precondition.

The reader has a right to be concerned at this point about the combinatorial explosion that may ensue as a result of applying AG1. As it now stands, this rule can introduce into the set of possible goals all the effects of any action that has on its precondition list any proposition already inferred to be a possible goal. Human experts are not nearly so profligate in hypothesizing possible goals of their advice-seekers. Instead they make extensive use of their knowledge of the domain and of the goals people are likely to hold in it. Similary, the use of rules such as AG1 by an automated expert will have to be governed by a control strategy that is expectation-driven. The development of such a control strategy is currently being studied.

We can now trace the expert's solution of "How do I delete a Control-Z?" The axioms used are numbered as in Section 6.5.2. Recall that the expert's task is to prove constructively the following theorem:

$$F \wedge T \supset \exists p \exists \alpha [POS\ want(p) \wedge achieves(\alpha, p, w_0)],$$

where F are the facts about w_0 and T is the translation of the query. There are many ways to prove this theorem, and many starts at proofs that will fail. What is presented here is the proof that results in the answer given by the human expert.

The proof will be presented in several parts. First, as already described, the expert deduces directly from the query that $[mode(create) \wedge msg(x_0)]$ is a possible goal of the advice-seeker:

0.	$WantAction(A, delete(control\text{-}z)) = WantAction(A, undo(control\text{-}z))$	
1.	$WantAction(A, undo(control\text{-}z))$	T1'
2.	$T(w_0, mode(send))$	F1
3.	$T(w_0, msg(x_0))$	F2
4.	$\forall p \forall q [Believes(A, \forall w_1 [T(w_1, p) \supset achieves(\alpha, q, w_1)]) \supset Believes(A, \forall w_2 [T(w_2, q) \supset achieves(undo(\alpha), p, w_2)])]$	*undo* des.

5.	$Believes(A, \forall w \forall x[T(w, mode(create) \wedge msg(x)) \supset achieves(control\text{-}z, mode(send) \wedge msg(x))])$	given
6.	$Believes(A, \forall w \forall x[T(w, mode(send) \wedge msg(x)) \supset achieves(undo(control\text{-}z), mode(create) \wedge msg(x), w)])$	mp,4,5
7.	$Believes(A, T(w_0, mode(send) \wedge msg(x_0))$	given
8.	$Believes(A, achieves(undo(control\text{-}z), mode(create) \wedge msg(x_0), w_0))$	mp,6,7
9.	$WantAction(A, \alpha) \wedge Believes(A, achieves(\alpha, p, w_0)) \supset POS\ want(A, p)$	LR1
10.	$POS\ want(A, mode(create) \wedge msg(x_0))$	emp,1,8,9

Line 1 substitutes the equality of line 0 into the query translation. Lines 2 and 3 are just F. Lines 4-10 use the description of the action *undo* to conclude that the advice-seeker wants to return to a state identical to the one that held just prior to his typing Control-Z. They are self-explanatory, except for the origin of lines 5 and 7. Either these can come from a user model that, unless there is reason to believe otherwise, assumes the advice-seeker understands the effects of actions, or they can be considered as expressed in the advice-seeker's query. For this particular example, the latter explanation is not farfetched.

Having determined that $[mode(create) \wedge msg(x_0)]$ is one possible goal, the expert next applies the alternative goal rule AG1 to determine another possible goal. One advantage of being in create mode is that the user can add to his current message:

11.	$\forall p \forall q \forall \alpha[[POS\ want(p) \wedge \forall w[T(w,p) \supset achieves(\alpha, q, w)]] \supset POS\ want(A, q)]$	AG1
12.	$\forall w \forall x \forall y[T(w, mode(create) \wedge msg(x)) \supset achieves(insert(y), mode(create) \wedge msg(x \circ y), w)]$	D2
13.	$\forall y\ POS\ want(A, mode(create) \wedge msg(x_0 \circ y))$	mp,10,12,11

The expert now knows many more possible goals of the advice-seeker: for each character he could type, the advice-seeker may want to be in create mode with a current message consisting of x_0 and that character appended to the end of it.

The alternative-goal rule can apply iteratively to its own output. By instantiating y, the expert can select one of the possible goals derivable from line 13, and then, instantiating the second conjunct of its hypothesis with D2, apply AG1 again. At the end of several such applications, she knows that a possible goal of the advice-seeker is to be in create mode with the current message consisting of x_0 appended with a string of characters, which will be denoted by γ:

14.	$POS\ want(A, mode(create) \wedge msg(x_0 \circ y_1))$	∀-inst.,13
15.	$\forall y POS\ want(A, mode(create) \wedge msg(x_0 \circ y_1 \circ y))$	mp,14,12,11
	... (several more applications of AG1 and D2)	
16.	$POS\ want(A, mode(create) \wedge msg(x_0 \circ \gamma))$	mp,12,11

Having determined that a possible goal is being in create mode with message $x_0 \circ \gamma$, the expert again applies AG1, but this time instantiating the second conjunct of the hypothesis with D1:

17.	$\forall w \forall x[T(w, mode(create) \wedge msg(x)) \supset$ $achieves(control\text{-}z, mode(send) \wedge msg(x), w)]$	D1
18.	$POS\ want(A, mode(send) \wedge msg(x_0 \circ \gamma))$	mp,16,17,11

The expert now knows that yet another possible goal of the advice-seeker is to be in send mode, with the msg x_0 appended with some string of characters. She can then choose to accept this instantiation of p, and attempt to find a plan α to achieve it. The planning process makes use of the definition of concatenation of actions, described in footnote 7:

19.	$\forall w \forall x[T(w, mode(send) \wedge msg(x)) \supset$ $achieves(EDIT, mode(edit) \wedge text(x), w)]$	D5
20.	$achieves(EDIT, mode(edit) \wedge text(x_0), w_0)$	mp,2,3,19
21.	$\forall w \forall x \forall y[T(w, mode(edit) \wedge text(x)) \supset$ $achieves(insert(y), mode(edit) \wedge text(x \circ y), w)]$	D3
22.	$\forall y\ achieves([EDIT; insert(y)], mode(edit) \wedge text(x_0 \cdot y), w_0)$	mp,20,21
	... (several more applications of D3)	
23.	$achieves([EDIT; insert(y_1); \cdots; insert(y_n)],$ $mode(send) \wedge text(x_0 \circ \gamma), w_0)$	mp,22,21
24.	$\forall w \forall x[T(w, mode(edit) \wedge text(x)) \supset$ $achieves(control\text{-}x; control\text{-}z, mode(send) \wedge msg(x), w)]$	D4
25.	$achieves([EDIT; insert(\gamma); control\text{-}x; control\text{-}z],$ $mode(send) \wedge text(x_0 \circ \gamma), w_0)$	23,24

The expert has thus succeeded in her top-level proof: she has instantiated p to the possible goal of being in send mode with the current message x_0 appended with some string of characters, and she has instantiated α to a plan that achieves that goal, namely typing EDIT, inserting the string of characters (which, through knowledge not accounted for here, she describe to describe as "the rest of your message"), and then finally typing Control-X Control-Z. This instantiation of α is what she presents as an answer.

6.6. Summary and the Work Ahead

The principal claim of this work is that goal inference is as critical to an automated system that provides expert advice as it is to an automated system that communicates in natural language. Providing expert advice is not a one-stage process in which the expert simply attempts to find a solution to the advice-seeker's stated problem. Rather she must first deduce what the problem might be, and only then look for a solution.

In the framework presented here, the advice-seeker's "problem" is cast as a goal formula that he wants to hold. To deduce that formula, the expert needs to do two things. First, unless the advice-seeker directly expresses some goal formula in her query, the expert must relate the query to a possible goal. Two types of rules were introduced to accomplish this: linking rules and action descriptions. ***Linking rules*** are general rules that relate the form of a query to the form of a goal.[14] One example of a linking rule asserts that, if a query requests that some action be performed, then a possible goal is what the advice-seeker believes to be the result of that action. ***Action description definitions*** are specific rules that capture people's commonly held beliefs about actions. It will be interesting to see, as this work continues, how much of the expert's knowledge can be represented by linking rules and how much by action descriptions.

Once the expert has linked the query to a possible goal, she may then have to determine how it might be related to other possible goals of the advice-seeker. ***Alternative-goal rules*** were introduced for this: the one described asserts that if some formula is a possible goal and if that formula is a prerequisite for some action, then the result of that action may also be a goal.

Figure 6.6-1 diagrams the process an expert infers the set of possible goals and then generates an answer.

Some first attempts at formulating the inference rules needed by the expert have been presented. However, their specifics are not hard and fast. Refining and developing the linking rules, action descriptions, and alternative goal rules will be a primary focus of the continuation

[14]The "form of a query" should be taken to refer not to a syntactic form, but rather to the answers to such questions as: does it express an action or a formula?; a conjunction or disjunction?

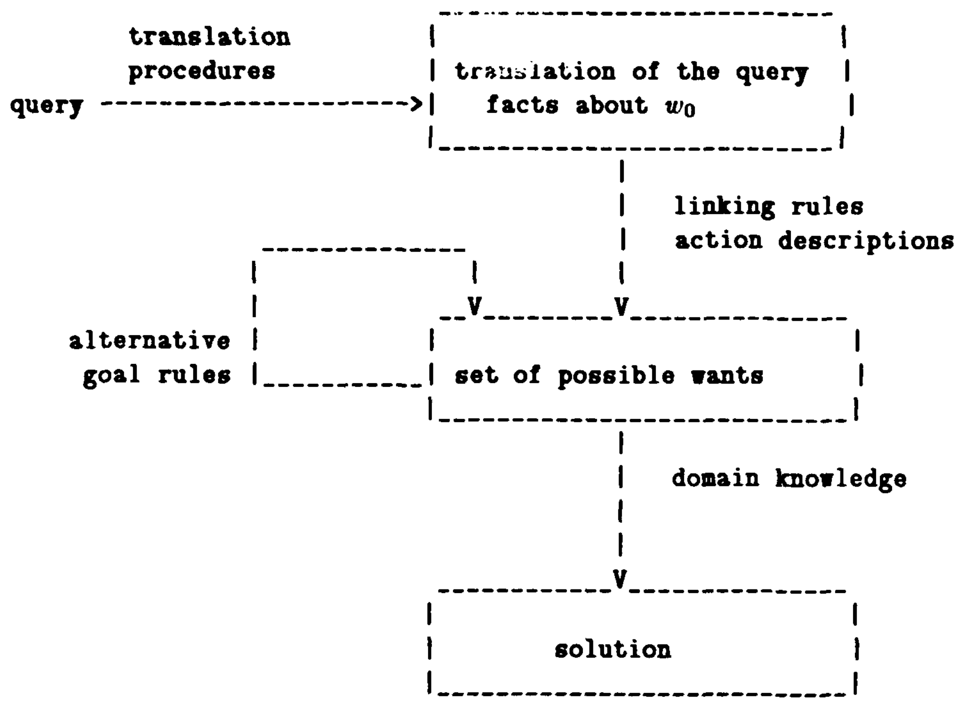


Figure 6.6-1: Generating an Expert Answer

of this work. One idea that merits consideration is whether possible-world semantics is the best representation to exploit. Alternative representations under consideration include the syntactic approach that Konolige [62] uses in his treatment of belief and the "situation semantics" being developed by Barwise and Perry [5]. It will also be interesting to develop the notion of critical frame-propositions, which were discussed in reference to the *achieves* predicate.

Finally, as was mentioned in Section 6.5.3 there has been no discussion in this work about a control strategy for the deduction system. Without a control strategy, it would be computationally infeasible for the expert to explore the extremely large number of inferrable possible goals. Fortunately, goal inference is not done in a vacuum: experts have knowledge as to what goals advice-seekers are likely to have. Allen and Perrault [2] term this knowledge the expert's *expectations*. Their system reasons in two directions at once, working both forward from what is expressed in the query, and backward from the expectations. It will be a major aim, as this work continues, to incorporate into it an account of the expert's knowledge of likely goals.

Much work remains to develop a complete model of the use of goal inference to generate appropriate expert answers. This work has established the need for such a model and, at the same time, has developed a framework for its construction.

7. The Role of Logic in Knowledge Representation and Commonsense Reasoning

This section was written by Robert Moore.

7.1. Introduction

In his AAAI presidential address, Allen Newell presented his view of the role that logic ought to play in representing and reasoning with commonsense knowledge [95]. Probably the most concise summary of that view is his proposition that "the role of logic [is] as a tool for the analysis of knowledge, not for reasoning by intelligent agents" [95, p. 16]. What we understand Newell to be saying is that, while logic provides an appropriate framework for analyzing the meaning of expressions in representation formalisms and judging the validity of inferences, logical languages are themselves not particularly good formalisms for representing knowledge, nor is the application of rules of inference to logical formulas a particularly good method for commonsense reasoning.

As to the first part of this position, we could not agree more. Whatever else a formalism may be, at least some of its expressions must have *referential semantics* if the formalism is really to be a representation of *knowledge*. That is, there must be some sort of correspondence between an expression and the world, such that it makes sense to ask whether the world is the way the expression claims it to be. To have knowledge at all is to have knowledge[1] that the world is one way and not otherwise. If one's "knowledge" does not rule out any possibilities for how the world might be, then one really does not know anything at all. Moreover, whatever AI researchers may say, examination of their actual practice reveals that they do rely (at least informally) on being able to provide referential semantics for their formalisms. Whether we are dealing with conceptual dependencies, frames, semantic networks, or what have you, as soon as we say that a particular piece of structure represents the assertion (or belief, or knowledge)

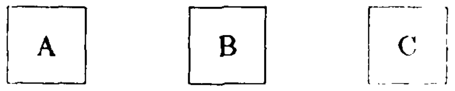
[1]or at least a belief; most people in AI don't seem overly concerned about truth in the actual world.

that John hit Mary, we have hold of something that is *true* if John did hit Mary and *false* if he didn't.

Now, mathematical logic (especially model theory) is simply the branch of mathematics that deals with this sort of relationship between expressions and the world. If one is going to provide an analysis of the referential semantics of a representation formalism, then, *a fortiori*, one is going to be engaged in logic. As Newell puts it [95, p. 17], "Just as talking of *programmerless* programming violates truth in packaging, so does talking of a *nonlogical* analysis of knowledge." It may be objected that we and Newell are overgeneralizing in defining logic so broadly as to include all possible methods for addressing this issue, but the fact remains that the only existing tools for this kind of semantic analysis have come from logic. We know this view is very controversial in AI, but will not argue the point any further for two reasons. First, it has already been argued quite eloquently by Pat Hayes [41], and, second, we want to go on to those areas where we *disagree* with Newell.

The main point on which we take issue with Newell is his conclusion that logical languages and deductive inference are not very useful tools for implementing (as opposed to analyzing) systems capable of commonsense reasoning. Newell does not present any real argument in support of this position, but instead says [95, p. 17] "The lessons of the sixties taught us something about the limitations of using logics for this role." In my view, Newell has seriously misread the lessons of the sixties with regard to this issue.

It appears to me that a number of important features of commonsense reasoning can be implemented *only* within a logical framework. Consider the following problem, adapted from Moore [89, p. 28]. Three blocks, A, B, and C, are arranged as shown:



A is green, C is blue, and the color of B is unstated. In this arrangement of blocks, is there a green block next to a block that is not green? It should be clear with no more than a moment's reflection that the answer is "yes." If B is green, it is a green block next to the nongreen block C; if B is not green then A is a green block next to the nongreen block B.

How is a person able to solve this problem? What sort of reasoning mechanisms are required? At least three distinctly "logical" factors seem to be involved: (1) the ability to see that an existentially quantified proposition is true, without knowing exactly which object makes it true, (2) the ability to recognize that, for a particular proposition, either it or its negation must be true, and (3) the ability to reason by cases. So far as we know, none of these abilities is possessed by any AI system not explicitly based on formal logic. Moreover, we would claim that, in a strong sense, these issues can be addressed only by systems that are based on formal logic.

To justify this claim we will need to examine what it means to say that a system uses a logical representation or that it reasons by deductive inference. Then we will try to re-evaluate what was actually shown by the disappointing results of the early experiments on problem-solving by theorem-proving, which we must do if the arguments presented here are correct and if there is to be any hope of creating systems with commonsense reasoning abilities comparable to those possessed by human beings.

7.2. What is a Logical Representation?

The question of what it means to use a logic for representing knowledge in a computer system is less straightforward than it might seem. In mathematics and philosophy, a logic is a language—i.e., a set of formulas—with either a formal inference system or a formal semantics (or both).[2] To use a logic in a computer system, we have to encode those formulas somehow as computer data structures. If the formulas are in "Cambridge Polish" notation, e.g.,

(EVERY X (IMPLIES (MAN X) (MORTAL X)))

we may be tempted to assume that the corresponding LISP S-expression must be the data structure that represents the formula in the computer. This is in fact the case in many systems, but using more sophisticated data structures certainly does not mean that we are not implementing a logical representation. For example, Sickel [120] describes a theorem-

[2]For example, for several decades there were formal inference systems for modal logic [51], but no semantics; Montague's intensional logic [88] has a formal semantics, but no inference system.

proving system in which a collection of formulas is represented by a graph, where each node represents a formula, and each link represents a possible unification (i.e., pattern match) of two formulas, with the resulting substitution being stored on the link. Furthermore, Sickel notes that the topology of the graph, plus the substitutions associated with the links, carries all the information needed by the theorem-prover—so the actual structure of the formulas is not explicitly represented at all!

This example suggests that deficiencies attributed to logical representations may be artifacts of naive implementations and do not necessarily carry over when more sophisticated techniques are used. For instance, one of the most frequently claimed advantages of semantic nets over logic as a representation formalism is that the links in the semantic net make it easier to retrieve information relevant to a particular problem. Sickel's system (along with that of Kowalski [66]) would seem to be at least as good as most semantic net formalisms in this respect. In fact, it may even be better, since following a link in a semantic net usually does not guarantee that the subsequently attempted pattern match will succeed, while in Sickel's or Kowalski's system, it does.

Given that the relationship between a logical formula and its computer implementation can be as abstract as it is in Sickel's system, it seems doubtful to me that we could give any clear criteria for deciding whether a particular system *really* implements a logical representation. We think that the best way out of this dilemma is to give up trying to draw a line between logical and nonlogical representations, and instead ask what logical features particular representation formalisms possess. If we adopt this point of view, the next question to ask is what logical features are needed in a general-purpose representation formalism. My answer is that, at a minimum, we need all the features of first-order classical logic with equality.

Perhaps the most basic feature of first-order logic is that it describes the world in terms of objects and their properties and relations. We doubt that anyone in AI could really complain about this, as virtually all AI representation formalisms make use of these concepts. It might be argued that one needs more than just objects, properties, and relations as primitive notions, but it should be kept in mind that first-order logic places no limits on what can be regarded as an object. Times, events, kinds, organizations, worlds, and sentences—not just concrete physical objects—can all be treated as logical individuals. Furthermore, even if we decide we need "nonstandard" features such as higher-order or intensional operators, we can still incorporate them within a logical framework.

For me, however, it is not the basic "metaphysical" notions of object, property, and relation that are the essential features of logic as a representation formalism, but rather the kinds of assertions that logic lets us make about them. Most of the features of logic can be seen as addressing the problem of how to describe an incompletely known situation. Specifically: existential quantification allows us to say that something has a certain property without having to know which thing has that property. Universal quantification allows us to say that everything in a certain class has a certain property without having to know what everything in that class is. Disjunction allows us to say that at least one of two statements is true without having to know which statement is true. Negation allows us to distinguish between knowing that a statement is not true and not knowing that it is true. Finally, logic lets us use different referring expressions without knowing whether they refer to the same object, but provides us with the equality predicate to assert explicitly whether or not they do.

One criticism of logic has been not that the above features are unnecessary or harmful, but rather that logic lacks some other essential feature—for instance, the ability to express control information. This was the basis of the early MIT-led criticism of theorem-proving research (e.g., [133, Chapter 6]), which was, we believe, largely justified. This sort of problem, however, can be addressed and, in fact, has been by Hayes [40], McDermott [86], Kowalski [67], and Moore [89] by extending logic in various ways (see Section 7.3), rather than by throwing it out and starting over. Moreover, the criticism quickly turned into a much more radical attack on any use of logic or deduction at all in AI [44, 45], [87, Appendix]. That assault, in my view, was tremendously detrimental to serious research on knowledge representation and commonsense reasoning and represents the position we primarily want to argue against.

The major reason we regard the features of first-order logic as essential to any general-purpose representation formalism is that they are applicable to expressing knowledge about *any* domain. That is, it doesn't really matter what part of the world we are talking about; it always may be the case that we have only partial knowledge of a situation and we need some of these logical features to express or reason with that knowledge. This can be seen in the example presented in Section 7.1. Reasoning about the position and color of blocks is certainly no more inherently logical than reasoning about anything else. The logical complexity of the problem comes from the fact that we are asked whether any blocks satisfy a given condition, but not which ones, and that we don't know the color of the middle block. If we had a complete description of the situation—if we were told the color of the middle block—we could just "read off" the answer to the question from the problem description without doing any reasoning at all.

Similar situations can easily arise in more practical domains as well. For instance, in determining a course of treatment, a physician may not need to decide between two possible diagnoses, either because the treatment is the same in either case or because only one of the two is treatable at all. Now, as far as we know, none of the inference methods currently being used in expert systems for medical diagnosis is capable of doing the sort of general reasoning by cases that ultimately justifies the physician's actions in such situations. Some systems have *ad hoc* rules or procedures for these special cases, but the creators of the systems have themselves had to carry out the relevant instances of reasoning by cases, because the systems are unable to. But this means that, in any situation the system designers failed to anticipate, the systems will fail if reasoning by cases is needed. It seems, though, that the practical utility of systems capable of handling only special cases has created a false impression that expert systems have no need for this kind of logic.

To return to the main issue, we simply do not know what it would mean for a system to use a nonlogical representation of a disjunctive assertion or to use a nonlogical inference technique for reasoning by cases. It seems to me that, to the extent any representation formalism has the logical features discussed above, it *is* a logic, and that to the extent a reasoning procedure takes account of those features, it reasons deductively. It is conceivable that there might be a way of dealing with these issues that is radically different from current logics, but it would still be *some* sort of logic and, in any event, at the present time none of the systems that are even superficially different from standard logics has any way of dealing with them at all.

Furthermore, the idea that one can get by with only special-purpose deduction systems doesn't seem very plausible to me either. No one in the world is an expert at reasoning about a block whose color is unknown between two blocks whose color is known, yet anyone can see the answer to the problem in Section 7.1. Intelligence entails being able to cope with novelty, and sometimes what is novel about a situation is the logical structure of what we know about it.

7.3. Why Did Early Experiments Fail?

Logic's bad reputation in AI circles for the past decade or so stems from attempts in the late 1960s to use general-purpose theorem-proving algorithms as universal problem-solvers. The idea was to axiomatize a problem situation in first-order logic and express the problem to be solved as a theorem to be proved from the axioms, usually by applying the resolution method developed by Robinson [109]. The results of these experiments were disappointing. The difficulty was that, in the general case, the search space generated by the resolution method grows exponentially (or worse) with the number of formulas used to describe a problem, so that problems of even moderate complexity could not be solved in reasonable time. Several domain-independent heuristics were proposed to try to deal with this issue, but they proved too weak to produce satisfactory results.

The lesson that was generally drawn from this experience was that any attempt to use logic or deduction in AI systems would be hopelessly inefficient. But, if the arguments made here are correct, there are certain issues in commonsense reasoning that can be addressed *only* by using logic and deduction, so we would seem to be at an impasse. A more careful analysis, however, suggests that the failure of the early attempts to do commonsense reasoning and problem-solving by theorem-proving had more specific causes that can be attacked without discarding logic itself.

We believe that the earliest of the MIT criticisms was in fact the correct one: that there is nothing particularly wrong with using logic or deduction per se, but that a system must have some way of knowing which inferences it should make out of the many possible alternatives. A very simple, but nonetheless important instance of this is deciding whether to use implicative assertions in a ***forward-chaining*** or ***backward-chaining*** manner. The deductive process can be thought of as a bidirectional search, partly working forward from premises to conclusions, partly working backward from goals to subgoals, and converging somewhere in the middle. Thus, if we have an assertion of the form $P \supset Q$, we can use it to generate either the assertion Q, given the assertion P, or the goal P, given the goal Q.

Some early theorem-proving systems utilized every implication both ways, leading to highly redundant searches. Further research produced more sophisticated methods that avoid some of these redundancies. Eliminating redundancies, however, creates choices as to which way assertions are to be used. In the systems that attempted to use only domain-independent control heuristics, a uniform strategy had to be imposed. Often the strategy was to use all assertions only in a backward-chaining manner, on the grounds that this would at least guarantee that all the inferences drawn would be relevant to the problem at hand.

The difficulty with this approach is that the question of whether it is more efficient to use an assertion for forward or backward chaining can depend on the specific form of that assertion. Consider, for instance, the schema

$$(\forall x)P(f(x)) \supset P(x)$$

Instances of this schema include such things as:

$$(\forall x)Jewish(mother(x)) \supset Jewish(x)$$

$$(\forall x)x' < y \supset x < y$$

That is, a person is Jewish if his or her mother is Jewish,[3] and a number x is less than a number y if the successor of x is less than y.

Suppose we were to try to use an assertion of this form for backward chaining, as most "uniform" proof procedures would. It would apply to any goal of the form $P(a)$ and produce

[3]We are indebted to Richard Waldinger for suggesting this example.

the subgoal $P(f(a))$. This expression, however, is also an instance of $P(x)$, so the process would be repeated, resulting in an infinite descending chain of subgoals:

$$\begin{array}{ll} GOAL: & P(a) \\ GOAL: & P(f(a)) \\ GOAL: & P(f(f(a))) \\ GOAL: & P(f(f(f(a)))) \\ & \vdots \end{array}$$

If, on the other hand, we use the rule for forward chaining, the number of applications is limited by the complexity of the assertion that originally triggers the inference:

$$\begin{array}{ll} ASSERT: & P(f(f(a))) \\ ASSERT: & P(f(a)) \\ ASSERT: & P(a) \end{array}$$

It turns out, then, that the efficent use of a particular assertion often depends on exactly what that assertion is, as well as on the context of other assertions in which it is embedded. Other examples illustrating this point are given by Kowalski [67] and Moore [89], involving not only the forward/backward-chaining distinction, but other control decisions as well.

Since specific control information needs to be associated with particular assertions, the question arises as to how to provide it. The simplest way is to embed it in the assertions themselves. For instance, the forward/backward-chaining distinction can be encoded by having two versions of implication—e.g., $P \rightarrow Q$ to indicate forward chaining and $Q \leftarrow P$ to indicate backward chaining. This approach originated in the distinction made in the programming language PLANNER [43] between antecedent and consequent theorems. A more sophisticated approach is to make decisions such as whether to use an assertion in the forward or backward direction *themselves* questions for the deduction system to reason about using "metalevel" knowledge. The first detailed proposal along these lines seems to have been made by Hayes [40], while experimental systems have been built by McDermott [86] and de Kleer et al. [20], among others.

Another factor that can greatly influence the efficiency of deductive reasoning is the exact way in which a body of knowledge is formalized. That is, logically equivalent formalizations can have radically different behavior when used with standard deduction techniques. For example, we could define *above* as the transitive closure of *on* in at least three ways:[4]

$$(\forall x, y) above(x, y) \equiv on(x, y) \vee (\exists z)[on(x, z) \wedge above(z, y)]$$
$$(\forall x, y) above(x, y) \equiv on(x, y) \vee (\exists z)[on(z, y) \wedge above(x, z)]$$
$$(\forall x, y) above(x, y) \equiv on(x, y) \vee (\exists z)[above(x, z) \wedge above(z, y)]$$

Each of these axioms will produce different behavior in a standard deduction system, no matter how we make such local control decisions as whether to use forward or backward chaining. The first axiom defines *above* in terms of *on*, in effect, by iterating upward from the lower object, and would therefore be useful for enumerating all the objects that are above a given object. The second axiom iterates downward from the upper object, and could be used for enumerating all the objects that a given object is above. The third axiom, though, is essentially a "middle out" definition, and is hard to control for any specific use.

The early systems for problem-solving by theorem-proving were often inefficient because axioms were chosen for their simplicity and brevity, without regard to their computational properties—a problem that also arises in conventional programming. To take a well-known example, the simplest LISP program for computing the n^{th} Fibonacci number is a doubly recursive procedure that takes $O(2^n)$ steps to execute, while a sligthly more complicated and less intuitively defined singly recursive procedure can compute the same function in $O(n)$ steps.

Kowalski [65] was perhaps the first to note that choosing among alternatives such as these involves very much the same sort of decisions as are made in conventional programming. In fact, he observed that there are ways to formalize many functions and relations so that the application of standard deduction methods will have the effect of executing them as efficient computer programs. These observations have led to the development of the field of "logic programming" [67] and the creation of new computer languages such as PROLOG [131].

[4]These formalizations are not quite equivalent, as they allow for different possible interpretations of *above* if infinitely many objects are involved. They are equivalent, however, if only a finite set of objects is being considered.

7.4. Summary and Conclusions

We have tried to argue that there is an important class of problems in knowledge representation and commonsense reasoning, involving incomplete knowledge of a problem situation, that so far have been addressed only by systems based on formal logic and deductive inference, and that, in some sense, probably can be dealt with only by systems based on logic and deduction. We have further argued that, contrary to the conventional wisdom in AI, the experiments of the late 1960s did not show that the use of logic and deduction in AI systems was necessarily inefficient, but only that better control of the deduction process was needed, along with more attention to the computational properties of axioms.

We would certainly not claim that all the problems of deductive inference can be solved simply by following the prescriptions given here. Further research will undoubtedly uncover as yet undiagnosed difficulties and, one hopes, their solutions. My objective here is to encourage consideration of these problems, which have been ignored for a decade by most of the artificial-intelligence community, so that at future conferences we may hear about their solution rather than just their existence.

8. The KLAUS Deduction System

This section was written by Mark Stickel.

The KLAUS deduction system is characterized by its use of nonclausal resolution as the basic inference rule, use of a connection graph to encode possible inference operations, demodulation and special unification for building-in equational theories, and heuristic search and control annotation for control.

8.1. Nonclausal Resolution

One of the most widely criticized aspects of resolution theorem proving is its use of clause form for wffs. The principal criticisms are

- Conversion of a wff to clause form may eliminate pragmatically useful information encoded in the choice of logical connectives (e.g., $\neg P \vee Q$ may suggest case analysis while the logically equivalent $P \supset Q$ may suggest chaining).
- Use of clause form may result in a large number of clauses being needed to represent a wff, as well as in substantial redundancy in the resolution search space.
- Clause form is difficult to read and not human-oriented.

The clausal resolution rule can be easily extended to general quantifier-free wffs [93, 81]. Proofs of soundness and completeness are in [93]. Where clausal resolution resolves on clauses containing complementary literals, nonclausal resolution resolves on general quantifier-free wffs containing atomic wffs (atoms) occurring with opposite *polarity*, which is determined by the parity of the number of explicit or implicit negations in whose scope the atom appears (positive polarity if even, negative polarity if odd). In clausal resolution, resolved-on literals are deleted and remaining literals disjoined to form the resolvent. In nonclausal resolution, all occurrences of the resolved-on atom are replaced by *false* (*true*) in the wff in which it occurs positively (negatively). The resulting wffs are disjoined and simplified by truth-functional reductions that eliminate embedded occurrences of ***true*** and ***false*** and optionally perform simplifications such as $\mathcal{A} \wedge \neg\mathcal{A} \rightarrow$ ***false***.

Definition 8.1. If $\mathcal{A}$ and $\mathcal{B}$ are ground wffs and C is an atom occurring positively in $\mathcal{A}$ and negatively in $\mathcal{B}$, then the result of simplifying $\mathcal{A}(C \leftarrow \textit{false}) \vee \mathcal{B}(C \leftarrow \textit{true})$, where $\mathcal{X}(\mathcal{Y} \leftarrow \mathcal{Z})$ is the result of replacing every occurrence of $\mathcal{Y}$ in $\mathcal{X}$ by $\mathcal{Z}$, is a *ground nonclausal resolvent* of $\mathcal{A}$ and $\mathcal{B}$.

It is clear that nonclausal resolution reduces to clausal resolution when the wffs are restricted to be clauses. In the general case, however, nonclausal resolution has some novel characteristics as compared with clausal resolution. It is possible to derive more than one resolvent from the same pair of wffs, even resolving on the same atom, if the atom occurs both positively and negatively in both wffs (e.g., atoms within the scope of an equivalence occur both positively and negatively). Likewise, it is possible to resolve a wff against itself.

The ground nonclausal resolution rule can be lifted to nonground wffs by renaming parent wffs apart and unifying sets of atoms from each parent, one atom of each set occurring positively in the first wff and negatively in the second. As with clausal resolution, only single atoms need be resolved upon if the resolution operation is augmented by a factorization operation that derives a new wff by instantiating a wff by a most general unifier of two or more distinct atoms occurring in the wff (regardless of polarity).

Factorization has not yet been implemented in the program. When two wffs are resolved upon a pair of atoms, all atoms instantiated to be the same as the instantiated resolved-on atoms are replaced by ***false*** or ***true***, but there is no effort to force additional atoms, by further instantiation, to be the same as the resolved-on atoms. Thus, only "obvious" factors are used. This is incomplete, but effective.

A nonclausal resolution derivation of ***false*** from a set of wffs demonstrates the unsatisfiability of the set of wffs. Nonclausal resolution is thus, like clausal resolution, a refutation procedure. Variants of the procedure that attempt to affirm rather than refute a wff are possible (e.g., see the variety of resolution rules in [81]), but are isomorphic to this procedure.

Although clause form is often criticized, use of nonclausal form has the disadvantage that most operations on nonclausal form are more complex than the same operations on clause form. The result of a nonclausal resolution operation is less predictable than the result of a clausal resolution operation. Clauses can be represented as lists of literals; sublists are appended to form the resolvent. Pointers can be used to share lists of literals between parent and resolvent [11]. With many simplifications such as $\mathcal{A} \wedge true \rightarrow \mathcal{A}$ and $\mathcal{A} \wedge \neg\mathcal{A} \rightarrow false$ being applied during the formation of a nonclausal resolvent, the appearance of a resolvent may differ substantially from its parents, making structure sharing more difficult.

For most forms of clausal resolution, an atom does not occur more than once in a clause. In nonclausal resolution, an atom may occur any number of times, with possibly differing polarity. In clausal resolution, every literal in the clause must be resolved upon for the clause to participate in a refutation. Thus if a clause contains a literal that is pure (cannot be resolved with a literal in any other clause), the clause can be deleted. This is not the case with nonclausal resolution; not all atom occurrences are essential in the sense that they must be resolved upon to participate in a refutation. For example, $\{ P \wedge Q, \neg Q \}$ is a minimally inconsistent set of wffs, one of which contains the pure atom P. A more complicated definition of purity involving this notion of essential occurrences must be used. The subsumption operation must also be redefined for nonclausal resolution to take account of such facts as the subsumption of $\mathcal{A}$ by $\mathcal{A} \wedge \mathcal{B}$ as well as the clausal subsumption of $\mathcal{A} \vee \mathcal{B}$ by $\mathcal{A}$.

[93, 81] suggest the extension of nonclausal resolution to resolving on nonatomic subwffs of pairs of wffs. For example, $P \vee Q$ and $(P \vee Q) \supset R$ could be resolved to obtain R. Resolving on nonatoms often permits significantly shorter and more readable refutations. However, there are several reasons for not doing this:

- It may be difficult to recognize complementary wffs. For example, $P \vee Q$ occurs positively in $Q \vee R \vee P$ and $\neg P \supset Q$.
- The effect of resolving a pair of wffs on nonatomic subwffs can be achieved by multiple resolution operations on atoms. Resolution on both atomic and nonatomic subwffs could result in redundant derivations.
- A connection-graph procedure would be complicated by the need to attach links to

logical subwffs (e.g., $P \vee Q$ in $Q \vee R \vee P$) and link inheritance would be further complicated since subwffs of a resolvent may have no parent subwffs (e.g., when $P \vee Q$ and $\neg P \vee R$ are resolved, the resolvent $Q \vee R$ is a subwff of neither parent). Similar complications arise if equality inferences are used that introduce new structure into the result.

Although the nonclausal resolution rule in general seems adequate as compared with the above proposed extension to matching on nonatomic subwffs, the handling of the equivalence relation in [93] is inadequate. In resolving $P \equiv Q$ and $(P \wedge R) \vee (\neg P \wedge S)$, it is possible to derive $Q \vee S$ and $\neg Q \vee R$, but not the more natural result of simply replacing P by Q. It is questionable whether handling the equivalence relation in nonclausal resolution without further extension is worthwhile in comparison with the representational advantages of negation normal form used in [3, 7]. Another difficulty with the equivalence relation is that it sometimes needs to be removed during skolemization. [82] provides extensions to nonclausal resolution that defer skolemization and permit equivalence relations to be retained longer.

8.2. Connection Graphs

Connection-graph resolution was introduced in [66]. It has the following advantages:

- The connection-graph refinement is quite restrictive. Many resolution operations permitted by other resolution procedures are not permitted by connection-graph resolution.
- The links associated with each wff function partially as indexing of the wffs. Effort is not wasted in the theorem prover examining the entire set of wffs for wffs that can be resolved against newly derived wffs.
- Links can be traversed by a graph-searching algorithm whereby each link traversal denotes a resolution operation. This can be done to plan a deduction without actually constructing it. This graph searching may resemble the searching performed for deduction in knowledge representation languages.

Connection-graph resolution is extended in a natural way to use the nonclausal resolution inference rule.

A connection graph is a set of wffs and a set of links that connect atoms occurring with positive polarity in one wff and negative polarity in the same or another wff. Performing the nonclausal resolution operation indicated by the link results in the production of a new connection graph with the resolved upon link eliminated and the nonclausal resolvent added. Roughly speaking, atoms of the nonclausal resolvent are linked only to atoms to which atoms of the parent wffs were linked.

Definition 8.2. Let $\mathcal{S}$ be a set of ground wffs. Let $\mathcal{L}$ be

$$\{\langle C, \mathcal{A}, \mathcal{B}\rangle \mid \mathcal{A}, \mathcal{B} \in \mathcal{S},\ \textit{atom } C \textit{ occurs positively in } \mathcal{A} \textit{ and negatively in } \mathcal{B}\}.$$

Then $\langle \mathcal{S}, \mathcal{L}\rangle$ is the *full connection graph* for $\mathcal{S}$.

Definition 8.3. Let $\mathcal{S}$ be a set of ground wffs and $\mathcal{L}$ be its connection graph. Let $\ell = \langle C, \mathcal{A}, \mathcal{B}\rangle$ be an element of $\mathcal{L}$ and $\mathcal{C}$ be the nonclausal resolvent $\mathcal{A}(C \leftarrow \textit{false}) \vee \mathcal{B}(C \leftarrow \textit{true})$. Let $\mathcal{S}$' be $\mathcal{S} \cup \{\mathcal{C}\}$. Let $\mathcal{L}$' be

$$\begin{aligned}
&\mathcal{L} - \{\ell\}\\
&\cup\{\langle E, \mathcal{C}, \mathcal{D}\rangle \mid \textit{atom } E \textit{ occurs positively in } \mathcal{C} \textit{ and } \langle E, \mathcal{A}, \mathcal{D}\rangle \in \mathcal{L} \textit{ or } \langle E, \mathcal{B}, \mathcal{D}\rangle \in \mathcal{L}\}\\
&\cup\{\langle E, \mathcal{D}, \mathcal{C}\rangle \mid \textit{atom } E \textit{ occurs negatively in } \mathcal{C} \textit{ and } \langle E, \mathcal{D}, \mathcal{A}\rangle \in \mathcal{L} \textit{ or } \langle E, \mathcal{D}, \mathcal{B}\rangle \in \mathcal{L}\}\\
&\cup\{\langle E, \mathcal{C}, \mathcal{C}\rangle \mid \textit{atom } E \textit{ occurs positively and negatively in } \mathcal{C} \textit{ and}\\
&\qquad \langle E, \mathcal{A}, \mathcal{A}\rangle \in \mathcal{L}, \langle E, \mathcal{B}, \mathcal{B}\rangle \in \mathcal{L}, \langle E, \mathcal{A}, \mathcal{B}\rangle \in \mathcal{L}, \textit{ or } \langle E, \mathcal{B}, \mathcal{A}\rangle \in \mathcal{L}\}.
\end{aligned}$$

Then the connection graph $\langle \mathcal{S}', \mathcal{L}'\rangle$ is derived from $\langle \mathcal{S}, \mathcal{L}\rangle$ by *ground nonclausal connection-graph resolution.*

A nonclausal connection-graph resolution refutation of an input set of wffs is a derivation of a set of wffs including *false* by nonclausal connection-graph resolution from the full connection graph of the input set of wffs.

Ground nonclausal connection-graph resolution can be extended to the nonground case by including in the links the unifier of the atoms they connect, keeping wffs renamed apart, and by including links between variants of the same wff (to allow a wff to directly or indirectly resolve against a variant of itself). Factorization must also be included. Either factors with appropriately inherited links must be added for each wff in the connection graph or special factor links can be used with link inheritance rules for both resolve and factor links after resolution and factorization operations.

The nonclausal connection-graph resolution procedure is sound and there is reason to believe it is complete. However, it has not yet been proved to be complete, and the history of proving completeness of connection-graph procedures for the simpler clausal case (see [7]) suggests it may be difficult.

One reason it is difficult to prove the completeness of the connection-graph procedure is that the link inheritance rules exclude some links that would be present if the connection graph were merely an encoding of all permitted resolution operations for ordinary resolution. Exactly which links are excluded depends on the order in which resolution operations are performed. The effect of connection-graph resolution is to impose the following restriction: if a pair of atoms in a pair of wffs is resolved upon, atoms derived (in later resolution operations) from the resolved-on atoms cannot be resolved against each other. For example, if a set of wffs includes $P \vee Q$ and $\neg P \vee \neg Q$, these two wffs can be resolved upon P and Q—resulting in tautologies that are discarded; after that, neither wff can be resolved with an atom descended from the other, even though doing so would not result in a tautology.

Connection-graph resolution procedures can possibly be incomplete by succeeding in finding refutations when links are resolved upon in some orders, but not others. For example, consider the combination of linear resolution and connection-graph resolution for clauses. Each is complete, but the combination is not. If linear connection-graph resolution is applied to $\{P \vee \neg Q, \neg P \vee \neg Q, Q\}$ with Q as top clause, depth-first search will find a refutation, but breadth-first search will not. This contrasts with the usual situation in which breadth-first search is "safe", always guaranteed to find a refutation if there is one. To see that it fails in this case, observe that after P and $\neg P$ are generated on the first level of breadth-first search, Q and $\neg Q$ have no links—and thus none of the three input clauses can be further resolved upon to lead to a refutation. P and $\neg P$ are linked, but cannot be resolved without violating the linear-resolution restriction.

A set of assertions in a connection graph can to some extent be regarded and treated as a semantic network—more so than the same set of assertions without the connection graph.

For example, the full connection graph for

$$\begin{array}{l} elephant(Clyde) \\ elephant(x) \supset mammal(x) \\ elephant(y) \supset color(y, gray) \\ mammal(z) \supset animal(z) \end{array}$$

would contain links between the following pairs of atoms

$$\begin{array}{ll} \ell_1. & \langle elephant(Clyde), elephant(x)\rangle \\ \ell_2. & \langle elephant(Clyde), elephant(y)\rangle \\ \ell_3. & \langle mammal(x), mammal(z)\rangle. \end{array}$$

Answers to such queries as "What color is Clyde?" and "Is Clyde an animal?" can be found by graph searching with minimal analysis of the assertions, by traversing the links in the connection graph. Such searching can be made more efficient by labeling the links (e.g., *isa* for ℓ_1 and ℓ_3, *hascolor* for ℓ_2). The semantic content of the set of assertions is still conveyed by the assertions themselves, but control information is provided to a graph-searching procedure by the link labels.

Similar comments could be made regarding any logical representation. However, the use of a connection graph in which all permissible remaining resolution operations are encoded in explicit links can yield greater efficiency by eliminating traversal of multiple paths to the same goal. For example, suppose ℓ_3 is resolved upon, resulting in the added assertion

$$elephant(w) \supset animal(w)$$

and the added link

$$\ell_4. \quad \langle elephant(Clyde), elephant(w)\rangle.$$

The link ℓ_3 is deleted. There is still only one path or proof that Clyde is an animal, since the absence of ℓ_3 blocks the path or proof $elephant(Clyde) \rightarrow mammal(Clyde) \rightarrow animal(Clyde)$.

Graph searching in the connection graph to determine taxonomic relations quickly is a simple illustration of the more general notion, extensively explored in [3, 120], of using graph searching to determine the existence of refutations. The ideas and techniques developed there are applicable to nonclausal connection-graph resolution. Connection-graph resolution appears to offer the following advantages over these other schemes:

- Although graph searching can be done in the connection-graph resolution procedure, [3, 120] do not allow for the actual formation of resolvents. If their techniques for graph search were adopted as a device for planning or quick refutation, connection-graph resolution could be regarded as a superset of these other methods.

- The actual formation of resolvents and the resulting change in the connection graph are useful for retaining information during a refutation, as well as for conveying information (about usage of wffs, etc.) from one refutation or assertion to the next. (Here it is assumed that the theorem prover is being used with an assertional database to which queries are posed and assertions occasionally added and deleted, as opposed to the usual situation in theorem proving in which there is no persistent assertional database, all axioms being presented anew for each proof.)

- Connection-graph resolution provides a convenient, albeit unsophisticated, means of interleaving matching complementary literals and adding new instances of assertions (if more than one ground instance of a wff is required), as compared with the separate processes of searching for a mating, and quantifier duplication if the search fails [3].

Of course, the argument in favor of performing only graph searching as in [3, 120] is that forming resolvents is expensive compared to traversing links, and the cost of creating and storing inherited links may be high.

A good system will probably have a mixture of resolution and graph searching, as in [8] for clausal connection-graph resolution. Graph searching is used in that system for look-ahead and to determine if a refutation exists within a certain number of steps. Simple graph searching is used (e.g., not looking for refutations in which wffs occur more than once), with the full complexity and completeness of connection-graph resolution in the background.

One problem with graph searching to find refutations is in assessing the effectiveness of the procedure. In ordinary resolution theorem proving, effectiveness can be evaluated in part by examining the number of clauses generated, retained, used in the refutation, and so forth. [8] states "Within this frame of reference it would be easy to design the 'perfect' proof procedure: the supervisor and the look-ahead heuristics would find the proof and then guide the system without any unnecessary steps through the search space." The amount of time used is a good measure for such a program, but should not be used to compare programs as there may be differences in the machines the programs run on and in the efficiency of the programs themselves (as opposed to the algorithms). In general, as [8] states, a measure incorporating both total time and space will be required, adding the further complication of evaluating time-space trade-offs.

The unification indexing provided by maintaining explicit links is augmented by indexing of all the atoms in the graph. This atom indexing is used in the process of integrating a new (user inputted) wff into the connection graph and is necessary for efficiency if there is a very large number of wffs already in the graph for which it must be determined if the new wff can be resolved with them; resolvents are linked according to the link inheritance rules which do not rely on the atom indexing. The atom index is also used to identify potential subsuming unit wffs although subsumption links [23] or demodulation could also be used for this purpose.

Variant elimination is used in addition to unit subsumption. Detection of variants depends on hash coding all the wffs in the graph with a hashing function that computes a hash code depending on all the symbols of the wff. The hashing function is insensitive to the order of the arguments of commutative functions, predicates, and connectives and the indices of variables in the wffs ($A \vee B$ and $B \vee A$ hash to the same value as do $P(x,y)$, $P(y,x)$, and $P(x,x)$), but ideally recognizes all other differences between wffs by assigning distinct hash codes. When adding a wff to the connection graph, it is first hashed and compared with all wffs previously in the connection graph with the same hash code to determine if it is a variant of one of them. If it is, the wff is not added to the graph.

8.3. Building-In Equational Theories

Building-in equational theories [103] can yield substantial improvements in the performance of a theorem-proving program by treating specially frequently recurring and costly to reason about concepts such as associativity, commutativity, identity, and inverse. Two mechanisms are presently employed to build-in equational theories: demodulation and special unification.

8.3.1 Demodulation

Demodulation [135] is the powerful technique of keeping all terms simplified with respect to a list of ordered equalities called demodulators. This reduces the size of the terms and, because equivalent terms may be simplified to the same term, facilitates subsumption. (A complete set of reductions [69, 69, 70, 50, 52] is essentially a list of demodulators that is *guaranteed* to simplify equivalent terms to the same term.) Demodulation can also be used as a programming mechanism for a variety of purposes [132] such as counting symbol occurrences in expressions, classifying expressions, etc.

Demodulation is extended beyond its conventional usage by permitting atomic formulas as well as terms to be simplified. In principle, demodulators can be written to perform arbitrary simplifications. term→term, atom→atom, and atom→truth-value simplifications are allowed. atom→term, etc. simplifications are illogical; atom→wff simplifications are useful (e.g., for expanding definitions) but presently unimplemented; it is expensive to check the applicability of wff→wff simplifications and the most useful cases can be anticipated and programmed directly (e.g.,$A \wedge \neg A \rightarrow$ *false*.)

atom→truth-value simplifications are particularly useful. For example, if $A \rightarrow$ *true* and $B \rightarrow$ *false* are demodulators, the clauses $\neg A \vee C$ and $B \vee C$ can be simplified to C and $A \vee C$ and $\neg B \vee C$ can be simplified to *true*—the effect is either that of a mandatory unit resolution operation plus subsumption of the parent clause or (using tautology elimination) of subsumption of the clause by a unit.

Such immediate, mandatory resolution and subsumption operations without retention of intermediate results accounts for much of the success of predicate calculus theorem proving using the Knuth-Bendix procedure [60, 69, 70, 50, 52] with a complete set of reductions for Boolean algebra [48].

Demodulation also provides a mechanism for implementing procedural attachment: the right-hand side of the demodulator specifies the compututation of a new expression for all expressions matching the left-hand side. This can be used to incorporate numerical computations. ([132] also uses demodulation to perform arithmetic operations, notably in counting demodulators.) It can also be used to attach code which imposes constraints, such as requiring a set of variables to have distinct values.

8.3.2 Special Unification

The incorporation of concepts such as associativity, commutativity, and idempotence into the unification algorithm has great potential for eliminating explicit equality reasoning, facilitating subsumption by recognizing nonidentical but equivalent terms, and generally reducing the size of the search space. Among the many special unification algorithms developed [106], the most pervasively useful special unification algorithms are those for associativity and/or commutativity with/without identity [75, 76, 122, 123, 126, 127], and it is these that have so far been implemented for this theorem prover. Building-in associativity and commutativity has usefulness well beyond the obvious mathematical usages such as handling symmetry of equality and associativity and commutativity of addition and mulltiplication.

A nonmathematical example of the usefulness of building associativity and commutativity into the unification algorithm, motivated by use of the theorem prover in natural-language understanding applications, is the use of $\neg T(K(x,w,n),p) \supset \neg T(K(Ker(x,y),w,n),p)$ written as part of a formulation of mutual knowledge [4]. It roughly states that if proposition p is not known by agent x, p is not common knowledge of agents x and y. If this implication is applied in a forward direction, it generates an infinite sequence of results *unless* Ker is treated as associative and commutative, in which case the first application of the axiom results in a formula which subsumes all the rest, eliminating an infinite branch in the search space.

Sets can be represented using the associative-commutative function *set* with identity: $\{a,b,c\}$ is represented by $set(a,b,c)$. The idempotence of sets can be handled either by (1) use of a demodulator $set(x,x,y) = set(x,y)$ which eliminates redundant elements or (2) declaring *set* to be idempotent. The former is easier and is preferable if in unifying $\{x,y\}$ and $\{a,b,c\}$ x and y are to be assigned disjoint sets of elements. However, if x and y must be assigned all sets of values such that their union is $\{a,b,c\}$, then *set* should be declared to be idempotent and use associative-commutative-idempotent unification with identity [76].

It is often necessary to select a single element of a set rather than (as in unifying $\{x,y\}$ and $\{a,b,c\}$) to decompose a set into two parts, neither of which is required to be a single element. In AI programming languages [43, 112] this problem is solved by making a distinction among variables: simple variables match single elements; fragment variables can match zero or more. If the set $\{a,b,c\}$ is encoded as $set(el(a), el(b), el(c))$, this distinction is unnecessary. If a variable is required to match a single element of the set, it is enclosed in *el*. For example, $x \in \{a,b,c\}$ can be expressed by $in(x, set(el(a), el(b), el(c)))$ and single element values for x can be nondeterministically selected by unification with the axiom $in(x, set(el(x), y))$.

8.4. Control

Two mechanisms have been used so far to control the process of searching for a refutation: heuristic search and control annotation. The heuristic search process assigns numerical scores to possible inference operations giving them a preference order. Limited control annotation is available to restrict ways in which specified assertions can be used. For example, special logical connectives are used to specify required forward chaining or backward chaining applications of implications.

8.4.1 Heuristic Search

A link scheduler is used to specify a refutation search strategy. When an assertion is added by the user, it is linked in the connection graph to all previous assertions. When a resolvent is added, it is linked to the other assertions according to the link inheritance rules. All such added links are examined by the link scheduler. Three outcomes are possible:

- The link is deleted. For example, analysis may show that resolving on the link would create a tautology or pure wff that could not be used in a refutation, whereupon the link can be deleted.
- The link is retained, but not scheduled. Thus the link can be inherited, but cannot be resolved upon (though its descendants might be). This is done when combining connection-graph resolution with other refinements of resolution, such as ordering restrictions and the special logical-connective restrictions described below.
- The link is scheduled. It is given a numerical score and placed in the link schedule. The theorem prover operates by repeatedly resolving on the best scored link in the schedule, creating the resolvent, and scheduling the added links.

Scheduling of the links is done after all the new links have been added, so that the link scheduler can act on such important facts as the number of links attached to an atom.

So far, only fairly simple evaluation functions have been used in the search control process. They are similar to those used in [125], being weighted sums of the deduction depth of the wff (a measure of the effort required to derive the wff) and the number of atoms in the wff (a measure of the additional effort that will be needed to complete a refutation using the wff). Performance is generally superior to that in [125]. In ordering restrictions, atoms are also evaluated according to how many links are connected to them, so that atoms with fewer links can be resolved upon preferentially. Not only is the immediate branching factor reduced, but there is also the prospect that the other atoms with more links will be instantiated and inherit fewer links when the resolution operation is performed. Interestingly, as was also noted in [125], there can be negative interactions among individually good ideas on search control. For example, a strong length-preference strategy and the strategy of resolving on an atom with the fewest links are somewhat inconsistent. When there are many assertions about some predicate—some short and specific, others long and general—the atom with the fewest links is likely to be linked only to long and general assertions. Resolving on it thus may result in long resolvents that would be given low preference by a strong-length preference strategy.

8.4.2 Control Annotation

Special logical connectives can be used to impose restrictions on the use of particular assertions. As in [89], the following connectives denote the following procedural interpretations of $A \supset B$:

- $A \rightarrow B$. If literal A is ever asserted, assert B (forward chaining).
- $B \leftarrow A$. To prove literal B, try to prove A (backward chaining). Since a refutation procedure is being used, this is interpreted as "permit the resolution, on literal B, between $A \supset B$ and any wff having support."
- $A \Rightarrow B$. If literal A is ever asserted, also assert B and, to prove $\neg A$, try to prove $\neg B$.
- $B \Leftarrow A$. To prove literal B, try to prove A and, if $\neg B$ is ever asserted, also assert $\neg A$.
- $A \supset B$ and $\neg A \vee B$. Unrestricted and equivalent.

The use of both nonclausal resolution and these special logical connectives gives this program some resemblance to natural deduction [9]. It represents an intermediate point between clausal resolution and natural deduction, with advantages of each. It differs from natural deduction, since, for example, a backward-chaining application of $A \supset B$ to C would result in $\neg A \vee C(B \leftarrow \mathit{true})$ rather than $C(B \leftarrow A)$ (with perhaps only a single instance of B replaced, requiring additional operations to replace the other occurrences). The latter expression may be more natural, but the former is more concise because all occurrences of B are eliminated and only a single instance of A is added. Heuristic search is used in a manner similar to the way it is employed in a clausal system [125] and in a natural-deduction system [128].

9. Theory Resolution: Building in Nonequational Theories

This section was written by Mark Stickel.

9.1. Introduction

Building theories into derived inference rules so that axioms of the theory are never resolved upon has enormous potential for reducing the size of the exponential search space commonly encountered in resolution theorem proving [109, 17, 78]. Plotkin's work on equational theories [103] was concerned with general methods for building in theories that are equational (i.e., theories that can be expressed as a set of either equalities or, by slight extension, equivalences of a pair of literals). This building in of equational theories consisted of using special unification algorithms and reducing terms to normal form. This work has been extended substantially, particularly in the area of development of special unification algorithms for various equational theories [106].

Not all theories that it would be useful to build in are equational. For example, reasoning about orderings and other transitive relations is often necessary, but using ordinary resolution for this is quite inefficient. It is possible to derive an infinite number of consequences from $a < b$ and $(x < y) \wedge (y < z) \supset (x < z)$ despite the obvious fact that no refutation based on just these two clauses is possible. A solution to this problem is to require that use of the transitivity axiom be restricted to occasions when either there are matches for two of its literals (partial theory resolution) or a complete refutation of the ordering part of the clauses to be refuted can be found (total theory resolution).

Another important form of reasoning in artificial intelligence applications addressed by knowledge representation systems [14] is reasoning about taxonomic information and property inheritance. One of our goals is to be able take advantage of the efficient reasoning provided by knowledge representation systems in this area by using the knowledge representation system as a taxonomy decision procedure in a larger deduction system. Combining such systems makes sense, since it relieves the general-purpose deduction system of the need to do taxonomic reasoning and, in addition, extends the power of the knowledge representation system towards greater logical completeness. Other researchers have also cited advantages of integrating knowledge representation systems with more general deductive systems [13, 108]. KRYPTON [12] represents an approach to constructing a knowledge representation system composed of two parts: a terminological component (the TBox) and an assertional component (the ABox). For such systems, theory resolution indicates in general how information can be provided to the ABox by the TBox and how it can be used by the ABox.

Building in nonequational theories differs from building in equational theories. Because equational theories are defined by equalities or equivalences, a term or literal can always be replaced by an equal term or equivalent literal. This is not the case for nonequational theories that are expressed in terms of implication rather than equivalence. If we build in a nonequational theory of taxonomic information that includes $Man(x) \supset Person(x)$, we would expect to be able to infer $Person(John)$ from $Man(John)$, but not $\neg Person(Mary)$ from $\neg Man(Mary)$ (i.e., replacement of $Man(x)$ by $Person(x)$ is permitted only in those cases in which Man occurs unnegated).

Nonequational theories may express conditional inconsistency of a pair of literals. For instance, $(x < y) \wedge (y < z) \supset (x < z)$ expresses the fact that $y < z$ and $\neg(x < z)$ are inconsistent only in the presence of $x < y$.

Theory resolution is a set of complete procedures for building in nonequational theories using decision procedures as components of a more general deduction system. Two forms are described. Total theory resolution employs a decision procedure that can determine inconsistency of any set of clauses using predicates of the theory and is quite restricted as to what inferences it will make. Partial theory resolution is less restricted as to what inferences it will make but requires much less of the decision procedure—making it more feasible, for example, to use knowledge representation systems as the decision procedure.

We will give definitions and completeness proofs for the ground case of theory resolution and definitions for the general case. Completeness for the general case follows directly from ground case completeness. We consider only clausal resolution here, but these results should be easily extendable to nonclausal resolution.

9.2. Total Theory Resolution

In building in a theory T, we are interested in ascertaining whether a set of clauses S is T-inconsistent (i.e., whether $S \cup T$ is inconsistent). If we have a decision procedure for T that is capable of finding minimally T-inconsistent subsets of clauses from any set of clauses using only predicates in T, then it can be applied to S with all literals having predicates not in T removed to create an inference rule (total theory resolution) that derives clauses containing no occurrrences of predicates that are referred to in T.

A theorem justifying such a rule of inference follows. In it, P is the set of predicates in T, S corresponds to $S \cup T$ above, and T is some subset of S_P. The decision procedure for T must determine T-inconsistency of sets of clauses from $S_P - T$ and W_P.

Theorem 9.1. Let S be a set of ground clauses and P be a set of predicates. Let S_P be the set of all clauses of S containing only predicate symbols in P. Let $S_{\bar{P}}$ be the set of all clauses of S containing only predicate symbols not in P. Let W be $S - S_P - S_{\bar{P}}$. Let W_P be the list of clauses C_i formed by restricting each clause in W to just the predicates in P. Let $W_{\bar{P}}$ be the list of clauses D_i formed by restricting each clause in W to just the predicates not in P. $W = \{ C_i \vee D_i \mid 1 \leq i \leq n \}$. Let X be the set of all clauses of the form $D_{i_1} \vee \cdots \vee D_{i_m}$ where $C_{i_1}, \ldots, C_{i_m}$ are all the clauses of W_P in a minimally inconsistent set of clauses from S_P and W_P. Then S is inconsistent if and only if $S_{\bar{P}} \cup X$ is inconsistent.

Proof: *If part. This proves the soundness of the rule.* Assume $S_{\bar{P}} \cup X$ is inconsistent. For every element of X, *false* is a logical consequence of S_P and some $C_{i_1}, \ldots, C_{i_m}$ from W_P. Therefore $D_{i_1} \vee \cdots \vee D_{i_m}$ is a logical consequence of S_P and W (derived, for example, by imitating a ground resolution derivation of *false* from S_P and $C_{i_1}, \ldots, C_{i_m}$ using $C_{i_1} \vee D_{i_1}, \ldots, C_{i_m} \vee D_{i_m}$ instead of $C_{i_1}, \ldots, C_{i_m}$). Because $S_{\bar{P}} \subseteq S$ and every element of X is a logical consequence of $S_P \cup W$ and $S_P \cup W \subseteq S$, if $S_{\bar{P}} \cup X$ is inconsistent, then so is S.

Only if part. This proves the completness of the rule. Assume S is inconsistent. Then either S_P is inconsistent, $S_{\overline{P}}$ is inconsistent, or the inconsistency of S depends at least partially on W.

Case 1. S_P is inconsistent. Then ***false*** $\in X$ and $S_{\overline{P}} \cup X$ is inconsistent.

Case 2. $S_{\overline{P}}$ is inconsistent. Then $S_{\overline{P}} \cup X$ is inconsistent.

Case 3. S_P and $S_{\overline{P}}$ are consistent. Because they have disjoint sets of predicates, $S_P \cup S_{\overline{P}}$ is also consistent. Then there is a minimally inconsistent set of clauses $S'_P \cup S'_{\overline{P}} \cup W'$ such that $S'_P \subseteq S_P$, $S'_{\overline{P}} \subseteq S_{\overline{P}}$, and $\emptyset \subset W' \subseteq W$. By completeness of A-ordered resolution [107, 123, 17, 78], there exists an A-ordered resolution refutation of this set with predicates in P preceding predicates not in P in the A-ordering. Included in the refutation is a set of clauses X' containing no predicates in P derived entirely from $S'_P \cup W'$. $S'_{\overline{P}} \cup X'$ is clearly inconsistent. When we look at the A-ordered derivations, it is apparent that each element of X' is of the form $D_{i_1} \vee \cdots \vee D_{i_m}$ derived from $S'_P \cup W'$ such that a subset of $S'_P \cup \{ C_{i_1}, \ldots, C_{i_m} \}$ is inconsistent where $\{ C_{i_j} \vee D_{i_j} \} \subseteq W'$. If this subset is minimally inconsistent then $D_{i_1} \vee \cdots \vee D_{i_m} \in X$. Otherwise $D_{i_1} \vee \cdots \vee D_{i_m}$ is still subsumed by (possibly identical to) an element of X. Because X contains each element of X' or an element that subsumes it, the inconsistency of $S_{\overline{P}} \cup X$ follows from the inconsistency of $S'_{\overline{P}} \cup X'$. ∎

Definition 9.2. Let $C_1, \ldots, C_m$ be nonempty clauses and $D_1, \ldots, D_m$ be clauses such that each $C_i \vee D_i$ is in S and every predicate in C_i is in theory T and no predicate in D_i is in theory T. Let $\sigma_{11}, \ldots, \sigma_{1n_1}, \ldots, \sigma_{m1}, \ldots, \sigma_{mn_m}$ be substitutions such that $\{ C_1\sigma_{11}, \ldots, C_1\sigma_{1n_1}, \ldots, C_m\sigma_{m1}, \ldots, C_m\sigma_{mn_m} \}$ is minimally T-inconsistent. Then $D_1\sigma_{11} \vee \cdots \vee D_1\sigma_{1n_1} \vee \cdots \vee D_m\sigma_{m1} \vee \cdots \vee D_m\sigma_{mn_m}$ is a ***total theory resolvent*** from S, using theory T.

Total theory resolution, plus ordinary resolution or some other semidecision procedure for first-order predicate calculus operating on clauses that do not contain predicates in T, is complete.

Example 9.3. Consider a theory of partial ordering ORD consisting of $\neg(x < x)$ and $(x < y) \wedge (y < z) \supset (x < z)$. A set of unit clauses in this theory is inconsistent if and only if it contains a chain of inequalities $t_1 < \cdots < t_n (n \geq 1)$ such that either $t_1 = t_n$ or $\neg(t_n < t_1)$ is one of the clauses. Total theory resolvents would include $F(b)$ from $(x < b) \vee F(x)$ and $F(a) \vee G(c)$ from $(x < b) \vee F(x)$, $(b < y) \vee G(y)$, and either $c < a$ or $\neg(a < c)$.

Thus the types of reasoning that are employable in the decision procedure can be quite different from and more effective (in its domain) than resolution.

There are limitations to the use of total theory resolution. The requirement that the decision procedure for the theory be capable of determining inconsistency of any set of clauses using predicates in the theory is quite strict. Reasoning about sets of clauses is probably an unreasonable requirement for such purposes as using a knowledge representation system as a decision procedure for taxonomic information, since such systems are often weak in handling disjunction. This tends to limit total resolution's applicability to building in mathematical decision procedures that handle disjunction. Incomplete restrictions of total theory resolution could still be usefully employed. For example, it may be easy for a system to decide inconsistency of sets of single literals (unit clauses), as above for *ORD* and maybe also in the case of taxonomic reasoning (note that taxonomic hierarchies can be expressed in the monadic predicate calculus for which there exists a decision procedure that could possibly be used in complete or incomplete theory resolution). Total theory resolution could then be used to resolve on only one literal from each clause.

Some care must be taken in deciding what theory T to build in so that the decison procedure does not have to decide too much. The theory must be capable of deciding sets of clauses that are constructed by using any predicates appearing in T. Thus, if we try to use total theory resolution to build in the equality relation with equality substitutivity (i.e., $x = y \supset (P(\cdots x \cdots) \supset P(\cdots y \cdots))$ for each predicate P), the decision procedure will have to decide all of S.

There may be a large number of T-inconsistencies that do not result in useful T-resolvents. It would be a worthwhile refinement to monitor the finding of T-inconsistent sets of clauses to verify that the substitutions applied do not preclude future use of the T-resolvent. This is like applying a purity check in A-ordered resolution.

A-ordered resolution slightly resembles total theory resolution. It permits resolution operations only on the atoms of each clause that occur earliest in a fixed ordering of predicates (the A-ordering). The A-ordering could place predicates of T before all others. A-ordered resolution differs from total theory resolution in that it assumes resolution (or hyperresolution) is to be used as the inference operation. It is thus inflexible, since it does not permit T to be built in except by resolution. Furthermore, total theory resolution creates a resolvent only from inconsistent set of clauses using predicates of T. A-ordered resolution is not so restrictive.

There is probably a useful relationship to be discovered between total theory resolution and the work on combining decision procedures [94, 119]. So far we have discussed only building in a single decision procedure, though the procedure could be repeated as long as the sets of predicates do not overlap. It is likely that we would want an extension of theory resolution that permits the sets of predicates to overlap at least in the case of the equality predicate. A difference between total theory resolution and the work on combining decision procedures is that the latter has been concerned primarily with decision procedures that do not have to instantiate their inputs, unlike our requirements for finding substitutions to make a set of clauses inconsistent.

9.3. Partial Theory Resolution

Partial theory resolution is a procedure for building in theories that requires a less complex decision procedure than total theory resolution; all it needs is a decision procedure that determines for any pair of literals a complete set of substitutions and conditions for the inconsistency of the literals.

Partial theory resolution will first be defined and proved complete for ground clauses. We will define a T-resolution operation that resolves on one or more literals from each input clause, like ordinary resolution without a separate factoring operation. We will then extend it to the general case, showing how T-resolvents can be computed by using only one literal at a time from each input clause.

There are two types of T-resolvents in partial theory resolution. If some set of literals of a clause is inconsistent with T, those literals can be removed from the clause to form a T-resolvent:

Definition 9.4. Let A be a nonempty ground clause and let C be a ground clause. Then C is a ***ground T-resolvent*** of $A \vee C$ if and only if $T \vdash \neg A$.

If, with T assumed, a set of literals of one clause is inconsistent with a set of literals of another clause under certain conditions, then T-resolvents can be formed as the disjunction of the other literals of the clauses and negated conditions for the inconsistency:

Definition 9.5. Let A and B be nonempty ground clauses and let C and D be ground clauses. Then $C \vee D \vee E$ where E is a ground clause is a ***ground T-resolvent*** of $A \vee C$ and $B \vee D$ if and only if $T \vdash \neg A \vee \neg B \vee E$ but not $T \vdash \neg A_i \vee E$ for any literal A_i in A or $T \vdash \neg B_j \vee E$ for any literal B_j in B. E is called the ***residue*** of matching A and B.

The residue E is a negated condition for the inconsistency of A and B because $T \vdash \neg A \vee \neg B \vee E$ is equivalent to $T \vdash \neg E \supset ((A \wedge B) \equiv \mathit{false})$. The restriction that neither $T \vdash \neg A_i \vee E$ nor $T \vdash \neg B_j \vee E$ assures us that all the literals of both of A and B are essential to the possible T-inconsistency of $A \wedge B$. T-resolution includes ordinary resolution because it is always the case that $T \vdash \neg A \vee \neg B \vee E$ if B is $\neg A$.

The soundness of ground T-resolution is obvious. T-semantic trees will be used to prove its completeness.

Definition 9.6. Let S be a set of ground clauses with set of atoms $\{A_1, \ldots, A_k\}$. Then a ***semantic tree*** for S is a binary tree with height k such that for each node n with depth $i\,(0 \leq i < k)$, n has two child nodes n_1 and n_2 at level $i+1$, the arc to n_1 is labeled by the literal A_{i+1}, and the arc to n_2 is labeled by the literal $\neg A_{i+1}$.

Each node n in a semantic tree provides a partial (or, in the case of terminal nodes, total) interpretation I_n for the atoms of S, assigning ***true*** to each atom A that labels an arc on the path from the root to n and assigning ***false*** to each atom A where $\neg A$ labels an arc on the path from the root to n.

Definition 9.7. Node n in a semantic tree for S *falsifies* clause $C \in S$ if and only if C is *false* in interpretation I_n.

Definition 9.8. A *T-semantic tree* is a semantic tree from which all nodes representing T-inconsistent truth assignments are removed.

Definition 9.9. Node n in a T-semantic tree for S *T-falsifies* clause $C \in S$ if and only if C is *false* in interpretation I_n taking account of T.

Definition 9.10. Node n is a *failure node* in a T-semantic tree for set of ground clauses S if and only if n T-falsifies a clause in S and no ancestor of n is a failure node.

Definition 9.11. Node n is an *inference node* in a T-semantic tree for set of ground clauses S if and only if both of n's child nodes are failure nodes.

Note that although in a T-semantic tree each nonterminal node may have either one or two child nodes, an inference node will always have two. If node n has only a single child node n_1 and n_1 T-falsifies a clause (and thus might be a failure node), then n also T-falsifies the clause. Thus, a failure node will never be the single child node of its parent.

Theorem 9.12. A set of ground clauses S is T-inconsistent if and only if every branch of a semantic tree $\mathcal{T}$ for S contains a failure node. Either the root node of $\mathcal{T}$ is a failure node or there is at least one inference node in $\mathcal{T}$.

Theorem 9.13. Let S be a T-inconsistent set of ground clauses and let $\mathcal{T}$ be a T-semantic tree for S. Then if the root node of $\mathcal{T}$ is not a failure node, there is a T-inconsistent set of ground clauses S' derivable from S by ground T-resolution such that $\mathcal{T}$ is a semantic tree for S' but has fewer failure nodes for S' than for S.

Proof: Since S is T-inconsistent and the root node of $\mathcal{T}$ is not a failure node, there must be at least one inference node n in $\mathcal{T}$. Then n's child nodes, n_1 and n_2, are both failure nodes. Let the clause T-falsified at n_1 be $A \vee C$ where nonempty clause A consists of all the literals not already T-falsified at n. Let the clause T-falsified at n_2 be $B \vee D$ where nonempty clause

B consists of all the literals not already T-falsified at n. Then n, or an ancestor of n if n is the single child node of its parent, is a failure node for $S' = S \cup \{C \vee D \vee E\}$ where E is a clause consisting of the negations of the literals labeling arcs above node n that were used in T-falsifying A and B. Because $T \vdash \neg E \wedge \neg A_i \supset \neg A$ and $T \vdash \neg E \wedge A_i \supset \neg B$ where A_i and $\neg A_i$ label the arcs to n_1 and n_2 respectively, $T \vdash \neg A \vee \neg B \vee E$ and $C \vee D \vee E$ is a ground clause T-resolvent of $A \vee C$ and $B \vee D$. $\mathcal{T}$ contains fewer failure nodes for S' than for S because n (or an ancestor of n) is a failure node instead of n_1 and n_2. ▮

Theorem 9.14. Ground clause T-resolution is complete.

Proof: Let S be a T-inconsistent set of clauses. Let $\mathcal{T}$ be a T-semantic tree for S. Then either the root node of $\mathcal{T}$ is a failure node, in which case the empty clause is derivable by ground T-resolution from the clause T-falsified at the root, or $\mathcal{T}$ contains an inference node, in which case completeness is assured by induction on the number of failure nodes, applying the previous theorem that uses ground T-resolution to add a clause that makes the inference node (or an ancestor of it) be a failure node. ▮

The ground T-resolution operation as defined above shares somewhat an undesirable feature of total theory resolution, i.e., that it demands too much of the decision procedure—in this case, requiring it to determine the possible inconsistency of two sets of literals (i.e., two clauses) instead of just two literals. This is easily remedied.

In the definition of ground T-resolvents with two parent clauses, let A be $A_1 \vee \cdots \vee A_m$ and let B be $B_1 \vee \cdots \vee B_n$. Then $T \vdash \neg A \vee \neg B \vee E$ is equivalent to all of $T \vdash \neg A_1 \vee \neg B_1 \vee E, \ldots,$ and $T \vdash \neg A_m \vee \neg B_n \vee E$ being true. Therefore, in computing T-resolvents, it is sufficient to determine possible T-inconsistency of single pairs of literals A_i and B_j with negated condition (residue) E_{ij} (i.e., $T \vdash \neg A_i \vee \neg B_j \vee E_{ij}$) and form T-resolvents of $A \vee C$ and $B \vee D$ as $C \vee D \vee E_{11} \vee \cdots \vee E_{mn}$. E_{ij} is a ***ground T-match*** of literals A_i and B_j. Note that this multiple pairwise matching is a substitute for a separate factoring operation.

We now extend T-matches and T-resolvents to the general (nonground) case.

Definition 9.15. Let A and B be two literals. Then $\langle E, \sigma \rangle$ where E is a clause and σ is a substitution is a *T-match* of A and B if and only if $T \vdash \neg A\sigma \vee \neg B\sigma \vee E$ but not $T \vdash \neg A\sigma \vee E$ or $T \vdash \neg B\sigma \vee E$.

A T-match of literals A and B specifies a substitution σ and condition $\neg E$ that make A and B be T-inconsistent.

We will give examples based on two theories:

- A taxonomic hierarchy theory TAX, including $Man(x) \supset Person(x)$.
- The partial-ordering theory ORD (defined previously).

Example 9.16. $\langle false, \{ w \leftarrow John \} \rangle$ is a TAX-match of $Man(John)$ and $\neg Person(w)$.

Example 9.17. $\langle a < c, \emptyset \rangle$ is an ORD-match of $a < b$ and $b < c$. But $\langle \neg(c < x) \vee (a < x), \emptyset \rangle, \langle \neg(c < x) \vee \neg(x < y) \vee (a < y), \emptyset \rangle, \ldots$ are also ORD-matches of $a < b$ and $b < c$. The notion of minimal complete sets of T-matches is defined to exclude these additional T-matches.

Definition 9.18. Let $M = \{ \langle E_1, \sigma_1 \rangle, \ldots, \langle E_n, \sigma_n \rangle \} (n \geq 0)$ be a set of T-matches of literals A and B. Then M is a ***complete set of T-matches*** of A and B if and only if for every T-match $\langle E, \sigma \rangle$ of A and B there is some T-match $\langle E_i, \sigma_i \rangle \in M$ and substitution θ such that $\sigma = \sigma_i \theta$ and $T \vdash E_i \theta \supset E$. M is a ***minimal complete set of T-matches*** of A and B if and only if M, but no proper subset of M, is a complete set of T-matches of A and B.

Example 9.19. $\{ \langle a < c, \emptyset \rangle \}$ is a minimal complete set of ORD-matches of $a < b$ and $b < c$. ORD-matches of the form $\langle \neg(c < x) \vee (a < x), \emptyset \rangle, \ldots$ have the property that $ORD \vdash (a < c) \supset [\neg(c < x) \vee (c < a)], \ldots$, and are therefore not in the minimal complete set of ORD-matches.

As in the ground case, single-parent and double-parent T-resolution operations are defined:

Definition 9.20. Let A be a literal and $A \vee C$ be a clause and let σ be a substitution such that $T \vdash \neg A\sigma$. Then $C\sigma$ is a ***T-resolvent*** of $A \vee C$.

Example 9.21. *Positive*(1) is an *ORD*-resolvent of $(x < 1) \vee Positive(x)$.

More than one T-inconsistent literal can be removed from a clause by performing single parent T-resolution repeatedly.

Definition 9.22. Let A and B be the nonempty clauses $A_1 \vee \cdots \vee A_m$ and $B_1 \vee \cdots \vee B_n$, let $A \vee C$ and $B \vee D$ be clauses, and let (E_{ij}, σ_{ij}) be T-matches of A_i and B_j. Then $C\sigma \vee D\sigma \vee E\sigma$ is a *T-resolvent* of $A \vee C$ and $B \vee D$ where σ is the most general combined substitution of $\sigma_{11}, \ldots, \sigma_{mn}$ and E is $E_{11} \vee \cdots \vee E_{mn}$.

Example 9.23. $\neg Robot(John)$ is a *TAX*-resolvent of $Man(John)$ and $\neg Person(w) \vee \neg Robot(w)$.

Example 9.24. $C(u) \vee D(u) \vee (a < c)$ is an *ORD*-resolvent of $(a < u) \vee C(u)$ and $(v < c) \vee D(v)$.

Further constraints on what T-resolvents can be inferred may be required for partial theory resolution to be really effective. For example, $a < b$ and $c < d$, which have no terms in common, have *ORD*-resolvents $\neg(b < c) \vee (a < d)$, $\neg(b < c) \vee (b < d)$, $\neg(b < c) \vee (a < c)$, $\neg(d < a) \vee (c < b)$, etc. If the first of these is actually used in a refutation, there must exist matches $b < c$ and $\neg(a < d)$ for its literals. It would be preferable to T-resolve these literals with $a < b$ and $c < d$ (e.g., T-resolve $a < b$ and $b < c$ deriving $a < c$, T-resolve $a < c$ and $c < d$ deriving $a < d$, and resolve that with $\neg(a < d)$) instead of directly T-resolving $a < b$ and $c < d$. We would impose the restriction that *ORD*-resolvents be derived only by resolving on pairs of literals that have a term in common.

There are two previous resolution refinements that resemble partial theory resolution: Z-resolution and U-generalized resolution.

Dixon's Z-resolution [21] is essentially partial theory resolution with the restriction that T must consist of a finite deductively closed set of 2-clauses (clauses with length 2). This restriction does not permit inclusion of assertions like $\neg Q(x) \vee Q(f(x))$, $\neg(x < x)$, or $(x < y) \wedge (y < z) \supset (x < z)$, but does permit efficient computation of T-resolvents (even allowing the possibility of compiling T to LISP code and thence to machine code).

Harrison and Rubin's U-generalized resolution [38] is essentially partial theory resolution restricted to sets of clauses that have a unit or input refutation. They apply it to building in the equality relation, developing a procedure similar to Morris's E-resolution [92]. The restriction to sets of clauses having unit or input refutations eliminates the need for factoring and simplifies the procedure (only a single literal of each parent must be used to create a T-resolvent), but otherwise seriously limits its applicability. No effort was made in the definition of U-generalized resolution to limit T-resolution by using minimal complete sets of T-matches.

Partial theory resolution is a procedure with substantial generality and power. Thus, it is not surprising that many specialized reasoning procedures can be viewed as instances of partial theory resolution, perhaps with additional constraints governing which partial theory resolvents can be inferred:

Where T consists of the equality axioms, T-resolution operations include paramodulation [134] (e.g., $P(b) \vee C \vee D$ can be inferred from $P(a) \vee C$ and $a = b \vee D$) and E-resolution [92] (e.g., $\neg a = b \vee C \vee D$ can be inferred from $P(a) \vee C$ and $\neg P(b) \vee D$).

Where T consists of ordering axioms, including axioms that show how ordering is preserved (such as $(x < y) \supset (P(x) \supset P(y))$ and $(x < y) \supset (x + z < y + z)$), T-resolution operations include Manna and Waldinger's program-synthetic special relation substitution rule (e.g., $P(b) \vee C \vee D$ can be inferred from $P(a) \vee C$ and $(a < b) \vee D$) and relation matching rule [82] (e.g., $\neg(a < b) \vee C \vee D$ can be inferred from $P(a) \vee C$ and $\neg P(b) \vee D$), which are extensions of paramodulation and E-resolution. T-resolution with ordering axioms is also similar to Slagle and Norton's reasoning about partial ordering [124]. Bledsoe and Hines's variable elimination [10] is a very refined method for reasoning about inequalities that can be viewed partly as partial theory resolution for inequality with added constraints on partial theory resolution operations. The ORD-resolvent $a < c$ of $a < b$ and $b < c$ is a variable-elimination-procedure chain resolvent only if b is a shielding term (nonground term headed by an uninterpreted function symbol). The variable elimination rule allows inferring ORD-resolvent $(a < b) \vee C$ from clause $(a < x) \vee (x < b) \vee C$ only if x does not occur in a, b, or C. The variable elimination rule more generally allows replacement of multiple literals $a_i < x$ and $x < b_j$ in a clause by literals $a_i < b_j$. This result is obtainable by partial theory resolution if we include the axiom $\neg(x < min(x,y))$ and a rule to transform $min(a_{i_1}, a_{i_2}) < b_j$ to $(a_{i_1} < b_j) \vee (a_{i_2} < b_j)$.

9.4. Conclusion

Theory resolution is a set of complete procedures for incorporating decision procedures into resolution theorem proving in first-order predicate calculus. Theory resolution can greatly decrease the length of refutations and the size of the search space, for example, by hiding lengthy taxonomic derivations in single TAX-matches and by restricting use of ordering axioms in ORD-matches. Total theory resolution can be used when there exists a decision procedure for the theory that is capable of determining inconsistency of any set of clauses using predicates of the theory. This may be a realistic requirement in some mathematical theorem proving. For example, a decision procedure for Presburger arithmetic (integer addition and inequality) might be adapted to meet the requirements for total theory resolution.

Partial theory resolution requires much less of the decision procedure. It requires only that conditions and substitutions for inconsistency of a single pair of literals be determinable by the decision procedure for the theory. This makes it feasible, for example, to consider use of a knowledge representation system as the decision procedure for taxonomic information. Partial theory resolution is also a generalization of several other approaches to building in nonequational theories.

10. A Prolog Technology Theorem Prover

This section was written by Mark Stickel.

10.1. Introduction

Prolog [131] is a powerful and versatile programming language based on theorem-proving unification and resolution operations.

The best Prolog implementations perform inferences at a rate that is often at least two orders of magnitude faster than theorem provers. Some of this disparity in speed can be accounted for by the fact that theorem provers often perform more complex inferences than Prolog (such as keeping results in fully simplified form and checking for subsumption).

However, one important reason for the higher speed of Prolog, compared with theorem provers, is the implementation. Given the present efficiency advantage of Prolog over theorem provers, and the fact that enormously more powerful Prolog machines are being contemplated (up to 10^9 logical inferences per second (lips) as opposed to the current best $10^4 - 10^5$ lips), it is worthwhile to examine the possibilities of adapting Prolog technology to theorem proving.

Prolog technology could be applied to theorem proving in a number of ways. To date, the most frequently used method of applying Prolog technology to theorem-proving problems is to substantially recode the problem in Prolog. Although performance may be high, this approach has significant limitations resulting from such implementation features of Prolog as unification without the "occurs check" and unbounded depth-first search. Also, the recoding process itself is time-consuming and error prone.

Prolog technology could be used in theorem proving by writing a theorem prover in Prolog, but this offers uncertain advantages in comparison with writing a theorem prover in any other language, such as LISP. Writing a theorem prover in Prolog would certainly result in a theorem prover whose inference operations are performed at a markedly lower rate than

Prolog's own, since several Prolog inference operations would have to be performed for each theorem-proving inference operation.

Prolog, as it now exists, almost meets the requirements for a complete theorem prover. Thus, we propose implementation of a slight extension of Prolog that permits full theorem proving directly. Direct modification of a Prolog interpreter, rather than coding a theorem prover in Prolog, preserves the speed of the Prolog interpreter by making extended Prolog operations be theorem-proving operations.

We are taking a fairly conservative approach to the extension of Prolog implementations for theorem proving. Simple additions to the Prolog interpreter should suffice to make the complete theorem prover—thus making the theorem prover easy to implement and similar to Prolog in its use. We retain such features of Prolog as the ordering of alternative inferences by statically ordering assertions in the database, the ordering of subgoals by statically ordering literals in assertions, and the cut operation. These features should be useful for programming a theorem prover just as they are for logic programming. Depth-first search, though bounded, will continue to be employed both for its comprehensibility and low storage requirements. Prolog also provides a convention for procedural attachment (built-in predicates) that should be useful in theorem proving as well.

We have two things in mind in presenting this design for a *Prolog technology theorem prover (PTTP)*. The first is that it employs highly efficient *Prolog technology* in its implementation. The second is that it is a *technology theorem prover* in the same way that TECH was a *technology chess player*[33]. It is a "brute force" theorem prover that relies less on detailed analysis than on high-speed execution of small logical steps. The capability of a PTTP would increase substantially as Prolog machine technology progresses.

We are currently experimenting with the concept of a PTTP that uses an extended Prolog interpreter (without all the Prolog built-in predicates) written in LISP with the same unification and substitution code employed in our other theorem-proving research. This allows experimentation with extended unification algorithms, but means that we do not yet have the efficiency of a true PTTP because the Prolog-style substitution representation is not being used.

10.2. A Minimal Prolog Technology Theorem Prover

Although Prolog uses unification and resolution for its matching and inference processes, it cannot be regarded as a full-fledged theorem prover. The deficiencies[1] lie in three areas:

- Unification without the occurs check
- Incomplete inference system
- Unbounded depth-first search strategy.

We will examine each of these problems in more detail and offer minimal solutions to them. The result will be the design of a minimal PTTP.

10.2.1 Unification

Prolog matching differs from the theorem-proving unification operation in only one respect: the absence in the former of the occurs check. In the theorem-proving unification operation, a variable is permitted to be instantiated to a term only if the variable does not occur in the term. This restriction eliminates the creation of infinite terms. The logical importance of this restriction is evident from the fact that without the occurs check it is possible to "prove" that $\forall x \exists y.P(x,y)$ implies $\exists y \forall x.P(x,y)$.[2] To prove this invalid result in Prolog, we match the skolemized form $P(sk2(y),y)$ of the goal $\exists y \forall x.P(x,y)$ and the skolemized form $P(x,sk1(x))$ of the assertion $\forall x \exists y.P(x,y)$. This match is successful without the occurs check.

It is clear that adding a straightforward occurs check to Prolog matching would impose unacceptable performance penalties on the operation of many logic programs. The lesser deduction depth and term complexity in typical theorem-proving applications would probably make it acceptable to add the occurs check. Furthermore, there are some easily verified

[1]While these are deficiencies from the standpoint of theorem proving, they are often assets in logic programming because they increase efficiency or comprehensibility of Prolog programs.

[2]I am indebted to Bob Moore for this observation.

circumstances in which the occurs check is unnecessary. When matching a goal with a clause head, it is unnecessary to perform the occurs check for the first variable binding; it is also unnecessary if the goal or the clause head has no variable occurring more than once. Use of the occurs check could be controlled by a run-time or compile-time switch.

An alternative approach to using the occurs check in each matching operation is a provision for checking at the completion of a proof to verify that no infinite term was created in the course of that proof.[3] Note that this approach requires that all bindings created during the course of a proof be available for checking upon its completion. This may not be the case for some Prolog implementations.

Either of these approaches should be easy to incorporate in an implementation of Prolog. There is a trade-off involved in the choice of approach. The first adds overhead to each unification but immediately blocks inferences using infinite terms. The second has little or no overhead for each unification but may permit many inferences to be drawn after an infinite term is created; these inferences could have been cut off by immediately using the occurs check.

10.2.2 Inference System

As is well known, the inference system used in Prolog is complete only for Horn sets of clauses, i.e., sets of clauses in which there is no more than one positive literal in each clause. We present a method of extending the Prolog inference system to a complete inference system that retains most of the character and efficiency of Prolog deduction.

In developing a PTTP, we should consider only those means for extending Prolog's inference system that permit highly efficient Prolog implementation techniques to be used. We observe that one of the most important reasons for the high speed of well-engineered Prolog implementations is the efficiency of their representation for variable substitutions. This representation is made possible both by the depth-first search strategy and by Prolog's use of a form of input resolution as its inference procedure.

[3] I first heard this suggestion from David Warren.

Two methods for handling substitutions are used in conventional resolution theorem proving. The simple method is to fully form resolvents by applying the unifying substitution to the parent clauses. This is far more expensive in both time and space than Prolog inference.

The second method is the ***structure-sharing*** approach [11], in which a resolvent is represented by the parents plus the unifying substitution. Whenever the resolvent must be examined (e.g., for printing or resolution with another clause), it is traversed with variables being implicitly replaced by their substitution values. This method consumes far less space than the simple method of fully forming the resolvents, but is still not very efficient in time, compared with Prolog. The reason for this relative inefficiency is clear.

In general resolution, a variable of an input clause may have more than one value per use of the clause in a deduction because the clause is implicitly reused whenever a descendant clause is used more than once. For example, if we resolve $P(x)$ and $\neg P(y) \vee Q(y)$, setting y to x, we obtain $Q(x)$. This resolvent can now be used twice to derive the empty clause from $\neg Q(a) \vee \neg Q(b)$. But this means that two instances of $P(x)$, $P(a)$ and $P(b)$, have been implicitly used in the proof, even though $P(x)$ was used explicitly only once. The substitution representation must accommodate these multiple variable values, whereas in Prolog the variable x can be implemented as a stack location containing [a pointer to] its single current value. The problem of multiple variable values does not occur in input resolution because derived clauses can only be used once. If it is assumed that each input clause is treated as a new clause as it is used, each variable will have only a single value in a single deduction.

This suggests that a good approach to building a PTTP is to employ a complete inference system that is an input procedure. Probably the simplest is the ***model elimination*** procedure [77, 78]. (Actually, what we are proposing here is more closely related to the problem-reduction-oriented MESON procedure [79, 78], but we will use the term model elimination (ME) because it is more familiar and the MESON procedure is derived from the ME procedure.)

The ME procedure requires only the addition of the following inference operation to Prolog to constitute a complete inference system for the first-order predicate calculus:

If the current goal matches the complement of one of its ancestor goals, then apply the matching substitution and treat the current goal as if it were solved.

This added inference operation is the ME *reduction* operation. The normal Prolog inference operation is the ME *extension* operation. The two together comprise a complete inference system.

An important thing to note is that this is a complete inference system that does not require the theorem-proving *factoring* operation. Basing an extension of Prolog on another form of model elimination, equivalent to SL-resolution [68], would require an additional factoring operation that would instantiate pairs of goals to be identical. Eder's Prolog-like interpreter for non-Horn clauses [22] also requires factoring. (However, we have not yet addressed Eder's concern regarding the type of search space redundancy that results in two proofs, not just one, of $\exists x.P(x)$ from $P(a) \vee P(b)$.)

For several reasons we regard factoring as an undesirable operation to add. Adding another inference operation requires further decision-making about how to order possible inference operations. The factoring operation, unlike extension and reduction, must use goals that no attempt has yet been made to solve—i.e., so far unexamined subgoals of the current or ancestor goals. The factoring operation, though necessary for completeness of many inference systems, has a tendency to instantiate goals excessively, thereby eliminating any possibility of solution.

The reduction operation is a form of reasoning by contradiction. If, in trying to prove P, we discover that P is true if Q is true (i.e.,$Q \supset P$) and also that Q is true if $\neg P$ is true (i.e., $\neg P \supset Q$), then P must be true. The rationale is that P is either true or false; if we assume that P is false, then Q must be true and hence P must also be true, which is a contradiction; therefore the hypothesis that P is false must be wrong and P must be true.

In Prolog, when a goal is entered, a choice point is established at which the alternatives are matching the goal with the heads of all the clauses and executing the body of the clause if the match is successful. In this extension of Prolog, we must also consider the additional alternatives of matching the entered goal with each of its ancestor goals. For each such successful match, we proceed in the same manner as if we had matched the goal with the head of a unit clause (a clause with an empty body).

In Prolog, when a goal is exited, the goal, instantiated by the current substitution, has been proved. In this extension of Prolog, when a goal is exited, all that has been proved is the instantiation of the goal disjoined with all the ancestor goals used in reduction operations in the process of "proving" the goal. Thus, in the example of proving P from $Q \supset P$ and $\neg P \supset Q$, expressed in Prolog by

```
p :- q.
q :- ¬p.
?- p.      ,
```

when goal q is exited, $P \vee Q$, not Q, has been proved. The top goal, when exited, has been proved; there are no ancestor goals whose negation could have been assumed in trying to prove the top goal.

One of the implementation requirements imposed by the addition of the reduction operation is that ancestor goals must be accessible. This precludes some optimizations such as a tail-recursive-call optimization that reuses the top stack frame when the next step is determinate and thus erases the current goal so that it cannot be used in a reduction operation.

There are two additional prerequisites for using this inference system. First, contrapositives of the assertions must be furnished. For each assertion with n literals, n Prolog assertions must be provided so that each literal is the head of one of the Prolog assertions. The order of the literals in the clause body can be freely specified by the user, as for ordinary Prolog assertions.

The second additional prerequisite relates to a feature of theorem proving that is absent in Prolog deduction: indefinite answers. Prolog, when provided with the goal $P(x)$, will attempt to generate all terms t such that $P(t)$ is definitely known to be true. In non-Horn clause theorem proving, there may be indefinite answers.

For example, consider proving $\exists x.P(x)$ from $P(a) \vee P(b)$. In our extension to Prolog, this can be expressed as

```
p(a) :- ¬p(b).
p(b) :- ¬p(a).
?- p(X).
```

This set of assertions and the described inference procedure are still insufficient to solve the problem because there is no term t for which it is definitely known that $P(t)$ is true. To solve problems with indefinite answers, it is necessary to add the negation of the query as another assertion (n assertions if the query has n literals).

In this example, addition of the Prolog assertion ¬p(Y) results in the finding of two proofs (one in which p(X) is matched with p(a) and ¬p(Y) is matched with ¬p(b), one in which p(X) matched with p(b) and ¬p(Y) is matched with ¬p(a)). The answer to the query is thus $P(a) \vee P(b)$, i.e., either $P(a)$ or $P(b)$ (or both) is true, but neither $P(a)$ nor $P(b)$ has been proved. In general, indefinite answers are disjunctions of instances of the query. One instance of the query is included for each use of the query in the deduction (the use of the query as the initial list of goals and each use of the negation of the query).

10.2.3 Search Strategy

Even if the problems of unification without the occurs check and an incomplete inference system are solved, or are irrelevant for a particular problem, Prolog is still unsatisfactory as a theorem prover because of its unbounded depth-first search strategy.

Consider the not untypical problem of proving that, in a monoid, if $x \times x$ is the identity element for every x, then $\times$ is commutative. This is often formulated in terms of the ternary predicate P, where $P(x, y, z)$ means $x \times y = z$ (this is quite consistent with Prolog relational programming style). The problem can then be expressed in Prolog by the following assertions and goal:

```
p(X,e,X).                                    % right identity
p(e,X,X).                                    % left identity
p(X,X,e).                                    % hypothesis that x × x = e
p(a,b,c).                                    % hypothesis that a × b = c
p(U,Z,W) :- p(X,Y,U), p(Y,Z,V), p(X,V,W).    % associativity rule 1
p(X,V,W) :- p(X,Y,U), p(Y,Z,V), p(U,Z,W).    % associativity rule 2
?- p(b,a,c).                                 % goal to prove that b × a = c
```

For this problem, Prolog's lack of the occurs check in unification and incomplete inference system do not matter, because no nonconstant function symbols appear and the set of clauses is a Horn set. However, Prolog will still fail to solve the problem because its unbounded depth-first search strategy will cause infinite recursion using the first associativity rule.

The minimal solution to the problem is to use bounded rather than unbounded depth-first search. Backtracking when reaching the depth bound will cause the entire search space, up to a specified depth, to be searched completely.

Because the search space size grows exponentially as the depth bound increases, assigning too large a depth bound for a particular problem may result in an enormous amount of wasted effort, and the amount of effort expended before discovering a proof will be highly dependent on the specified depth bound. The obvious solution to this problem is to run a PTTP with increasing depth bounds—first one tries to find a proof with depth 1, then 2, etc. We will call this the ***staged depth-first search strategy***. Because of the exponential growth of the size of the search space as the depth bound increases, the cost of searching all of levels $1, \ldots, n$ before first finding a proof at level $n + 1$ will probably not be unacceptably high relative to the cost of just searching at level $n + 1$.[4]

Rather then make all inferences up to level n, we should make only those that have some chance of resulting in a proof by level n. Because each as yet unsolved goal will require at least one inference step to solve it, we should not perform any inference step that would result in there being more unsolved goals than there are levels remaining before the depth bound is reached.

This approach has some other consequences for logic programming. The use of depth-bounded search changes the meaning of failure from "not provable" to "not provable within depth bound", thus requiring rejection or modification of the treatment of failure as negation.

[4]Assuming that the search space has a uniform branching factor b, $S(b,n) = b^n + b^{n-1} + \cdots + b^2 + b$ is the number of inferences made in exhaustively searching through level n and $SS(b,n) = b^n + 2b^{n-1} + \cdots + (n-1)b^2 + nb$ is the cumulative number of inferences made in exhaustively searching through level $1, 2, \ldots, n$. Then $S(b, n+1) = b^{n+1} + S(b,n) = b^{n+1} + SS(b,n) - SS(b,n-1)$ and $S(b,n+1) - SS(b,n) = b^{n+1} - SS(b,n-1) = b^n(b - \frac{1}{b} - \frac{2}{b^2} - \cdots - \frac{n-1}{b^{n-1}})$ implying that the cost of exhaustively searching through level $n+1$ usually greatly exceeds the accumulated costs of exhaustively searching through all of the previous levels.

The use of depth-bounded search with increasing depth bound also will cause side effects to be repeated, because deduction steps occurring in the level n search will be repeated in the level $n + 1$ search.

10.3. Refinements

10.3.1 Goal Acceptability

The ME procedure justifies the completeness of our extension of Prolog even if some goal states are disallowed. Let us call a goal currently being solved (either it or one of its subgoals is the current goal) an *open goal.* An *unopened goal* is a goal not yet open in the current deduction. A *closed goal* is a goal that has been exited in the current deduction.

Our extension of Prolog remains complete even if we allow the current goal to be failed under any of the following circumstances:

- Two unopened goals from the same clause are complementary
- A goal is identical to an ancestor goal
- A goal extended upon is complementary to an ancestor goal.

The first rule is justified because, in that situation, a tautologous instance of the clause is being used. Completeness is preserved if tautologous input clauses are not used. The second rule requires a more detailed justification, but in essence states that it is unnecessary to attempt to solve a goal while in the process of attempting to solve that same goal. The third rule merely affirms that it is unnecessary to attempt to solve a goal that is complementary to an ancestor goal by any means other than the reduction operation.

Because the search space in theorem proving is generally exponential, it is always worth considering criteria for failing goals, so that the exponentially many derivative deductions can be eliminated. However, the desire to cut off deductions must be balanced against the cost of applying the check to determine whether the present deduction is acceptable according to the criteria.

The ME procedure applicability tests enumerated above are very expensive to apply in a straightforward way. Because each inference operation is potentially capable of instantiating any goal, one of the conditions for unacceptability may become true for a pair of goals after any inference operation. Thus, after each inference operation we would have to check each pair of unopened goals from the same clause for complementarity, and each goal and its ancestor goals for identity and complementarity. The latter is $O(n^2)$, where n is the number of ancestor goals.

There are two solutions to the high cost of these applicability tests. The first is to develop an implementation that can perform these tests cheaply. One method would be to keep track of which pairs of goals could conceivably be instantiated to identity or complementarity and check only those pairs. However, this would result in a more substantial extension of Prolog than we presently want and would make it more difficult to adapt present Prolog implementations to be a PTTP.

The second solution is to restrict the applicability tests. First, we would eliminate the test for complementarity of unopened goals from the same clause. Besides saving the effort of performing the test, we eliminate the requirement for accessing unopened goals. Second, we suggest that the checking of a goal and its ancestor goals for identity and complementarity be restricted to the case where the goal is the current goal; this is done after instantiation by the substitution for the contemplated inference operation. Limited experience suggests that this single check is still quite successful in cutting off search at far less cost (linear in the number of ancestor goals) than the fuller check.

Another possible effort-saving restriction on the applicability tests would be to perform them less frequently than after every inference operation.

The previous theorem prover that most closely resembles a PTTP (in operation but not in implementation or speed) is an implementation of the ME procedure by Fleisig *et al* [26]. They concluded that ME was a competitive procedure; neither the ME theorem prover nor a unit preference and set-of-support resolution theorem prover they also developed strongly dominated the other for their examples. Their ME theorem prover uses full acceptability checking, that we consider likely to be too time-consuming, and a bounded depth-first search strategy. Unlike our staged depth-first search strategy, a single depth bound is given by the user, making performance very sensitive to the depth bound.

The Fleisig theorem prover also provides for cutoffs by allowing restrictions to be placed on the depth of function nesting, the number of open goals (or number of ancestor goals) in a deduction, the number of uses of particular clauses in a deduction, and the number of uses of clauses of specified length in a deduction. Such cutoffs can also be employed in a PTTP. Although they may ultimately be necessary to reduce the size of the exponential search space for difficult problems, we are somewhat wary of such cutoffs because they are sensitive parameters whose values are difficult to assign. To be useful, the cutoffs must be assigned small values, but not so small as to preclude all proofs. When there are many such parameters, there may be little guidance on which parameter values to alter to admit more inferences when no proof is found with one set of parameter values.

10.3.2 Extended Unification

It is sometimes quite useful to extend the unification algorithm. For example, building associativity and/or commutativity into the unification algorithm can result in significantly improved performance. Extended unification can also be used for helping to produce systems that reason effectively with equality, taxonomies, ordering, etc. (see Section 9). Kornfeld's work on building uses of equality into Prolog to support object-oriented programming is a further example [63]. Unlike his work however, full support for extended unification must accommodate the possible presence of multiple unifiers. This means additional alternatives at each choice point—alternative unifiers as well as alternative inferences. The clearest implementation of this would require that all alternative unifiers for an inference be tried before the next alternative inference is tried. It would also be useful to have an additional cut operation that cuts off alternative unifiers but not alternative inferences.

10.3.3 Operation Ordering

We retain the operation ordering of Prolog (solving subgoals from left to right; using clauses in order from the database) for familiarity, comprehensibility, and programmability. However, the addition of the reduction operation means that the reduction operation must be fitted in somewhere among the other operations. It must be decided whether reduction operations should be performed before, after, or interleaved with extension operations (e.g., after all extensions by unit clauses). This can be specified *a priori* or, perhaps, for each predicate P by including a clause "p :- reduce." in the procedure for P at the point where we wish reduction operations to be attempted (which could also make reduction optional). The order of reduction operations among themselves must also be decided—for example, whether to reduce by the shallowest or deepest ancestor goals first.

It is also worth pointing out that it is unnecessary to consider alternative inference operations if a reduction operation is possible where the two goals are already complementary (the empty substitution unifies their atoms) or if an extension operation by a unit clause instantiates only the clause and not the goal. Discarding alternative inference operations in these situations can save substantial effort.

The fullest possible benefit of this would be obtained if all reduction and unit extension operations were checked first to determine whether the current goal can be solved immediately without further instantiation before performing any inference operation that either further instantiates the current goal or adds subgoals. This suggests a two-pass procedure for attempting to solve a goal: checking ancestor goals for exact complementarity and for subsuming unit clauses; then, if that fails, performing the normal inference operations.

10.3.4 Additional Inference Operations

It is possible to consider adding more inference operations to a PTTP beyond the extension and reduction operations it minimally requires. We have already considered and rejected the idea of including a factoring operation.

A more useful operation to add may be the graph construction procedure *C-reduction* operation [118]. If the current goal matches a closed goal in the current deduction, the substitution can be applied and the current goal considered as solved, provided that all the ancestor goals used in reduction operations to solve the closed goal are also ancestors of the current goal. One difficulty in using this inference rule is that it adds redundancy to the search space. If the current goal can be solved by the C-reduction operation, it can also be solved by the same sequence of inference operations that was used to solve the C-reducing goal.

A strategy for using the C-reduction operation that is guaranteed not to increase the size of the search space is to apply the C-reduction operation only if the current goal is identical to the solved goal, so that no instantiation is required. Then all alternative inference operations that could be used to solve the current goal can be cut off, in the same way as was suggested for reduction by identical ancestor goals and extension by subsuming unit clauses.

10.4. Conclusion

We have presented the design of a minimal Prolog technology theorem prover and numerous possible refinements. Further experimentation will determine how worthwhile the concept is. Numerous questions, of course, remain. We have based our design on the ME procedure because that appears to be the procedure best suited to the use of present Prolog-style implementation. Is this the most effective procedure for us? How useful is it when viewed as a logic programming language? Will the lack of subsumption, equality reasoning, or other features impose too great a limit on its effectiveness? If necessary, how can we add such features while retaining the speed advantage?

In any case, production of a PTTP would result in a theorem prover capable of performing inferences at a far greater speed than before and offering prospects of even greater speed as Prolog machine technology progresses. It is surely a concept that is worth exploring.

11. Knuth-Bendix-Method Theorem-Proving Example

This section was written by Mark Stickel.

11.1. Introduction

We present here the solution of a challenge problem using the Knuth-Bendix complete sets of reductions method. The problem—to prove that, if $x^3 = x$ in a ring, then the ring is commutative—was offered as a challenge for theorem-proving programs by W.W. Bledsoe in 1977 [9]. The proof has the novel feature that all reasoning was forward reasoning with the program never having been told that the objective was proving $x \times y = y \times x$.

This is a substantial success for the Knuth-Bendix completion method that had already shown promise in solving less difficult problems such as completing sets of reductions for various algebras like free groups and rings. It further suggests that the Knuth-Bendix completion method is a very effective method for deriving consequences from equational theories whose equations can be treated as reductions. That a 21 step proof (not counting reduction steps) for this difficult problem could be found after generating only 135 reductions is quite impressive.

The technique used was the Knuth-Bendix completion method using associative-commutative unification for addition and incomplete associative unification for multiplication. The program attempted to find a complete set of reductions beginning with a complete set of reductions for free rings plus the reduction $x \times x \times x \rightarrow x$. The program predictably failed to complete the set of reductions. Commutativity of multiplication, a consequence of $x^3 = x$, prevents there being a complete set of reductions unless commutativity of multiplication is assumed. However, the program did discover the commutative equality $x \times y = y \times x$ in the attempt, thus proving the theorem. The program also later succeeded in proving the same result with comparable effort by using ordinary, rather than associative, unification for multiplication and building associativity in with the reduction $(x \times y) \times z \rightarrow x \times (y \times z)$.

We will not review here the Knuth-Bendix method or its extension using associative and/or commutative unification. The Knuth-Bendix method is described by Knuth and Bendix [60], Lankford [69, 70], and Huet [50, 49], among others. Associative and/or commutative unification are treated by Siekmann [121, 122], Livesey and Siekmann [75, 76], and Stickel [126, 127] and extension of the Knuth-Bendix method to incorporate associativity and/or commutativity is treated by Lankford and Ballantyne [71, 72, 73] and Peterson and Stickel [101, 102]. In addition to the publications cited above, Hullot [52] presents numerous examples of the use of the Knuth-Bendix method with and without special treatment of associativity and/or commutativity.

The two most important differences in the use of the Knuth-Bendix method for this problem, as compared to previous work by Peterson and Stickel [101, 102], are the use of cancellation laws to simplify reductions and a better pair-evaluation function to order matching reductions. These changes are described below.

11.2. Cancellation Laws

The most significant addition to the Knuth-Bendix method using associative and/or commutative unification that we made to solve the $x^3 = x$ ring problem is the use of cancellation laws to simplify derived equations. This addition made the solution feasible; we have so far failed to solve the $x^3 = x$ ring problem without the cancellation laws. It is certain that the effort required to do so would greatly exceed that used with the cancellation laws. We expect that cancellation laws can be widely used in the Knuth-Bendix method in the future to substantially accelerate convergence of complete sets of reductions.

To use the cancellation laws, we add the reductions $x + y = x \rightarrow y = 0$ and $x + y = x + z \rightarrow y = z$ that may be applicable to the entire derived equality, not just one of its subterms as is the case for all the other reductions.

With the single exception of the additive identity reduction $x+0 \rightarrow x$ that the cancellation laws are not permitted to reduce the cancellation laws never reduce an equality of nonidentical terms to an equality of identical terms. Thus, any critical pair from which a reduction can be derived can lead instead to a simpler reduction if a cancellation law is applicable. This simpler reduction is more powerful than the original because it, plus $x + 0 \rightarrow 0$, can reduce the original reduction to an identity.

Besides being more powerful, the simpler reduction has the further advantage that matching its left-hand side with the left-hand side of other reductions to generate new critical pairs will result in fewer, less complex equalities and thus create less work for the program.

11.3. Pair-Evaluation Function

In the Knuth-Bendix method, it is necessary to select which pair of reductions to match next to derive new critical pairs. If the selection algorithm is poor, the completion process may diverge, even though a complete set of reductions exists. There is no way of insuring that the completion process will not diverge unnecessarily, but selecting pairs with small combined left-hand sides works well in practice in preventing this difficulty.

Our implementation of the pair-selection process involves maintaining a list of all pending pairs sorted in ascending order according to an evaluation function. The evaluation function we have used in the past is simply the sum of the number of symbols in the two left-hand sides. However, the $x^3 = x$ ring problem is much more difficult than previous problems to which the Knuth-Bendix method has been applied, and this simple evaluation function was not adequate for easily solving it. An attempt to solve the $x^3 = x$ ring problem managed to get within one step of discovering $x \times y = y \times x$, but failed to select the right pair of reductions to match next after several days of computing.

The problem was the discovery of large numbers of reductions like $x^2yxyx^2yx \rightarrow xyx$. Such reductions have few symbols on the left-hand side and hence were given preference for matching. However, matching these reductions with other reductions (such as the distributivity reductions) often resulted in a large number equations that were slow to simplify. Simply counting symbols in the left-hand side of the reduction did not reflect the greater complexity (after simplification to sum-of-products form) of products when a variable is instantiated to a sum compared to the complexity of sums when a variable is instantiated. The solution is to use an evaluation function that gives less preference to products.

The evaluation function adopted to remedy this problem is $V(\lambda_1) + V(\lambda_2)$ where λ_1 and λ_2 are the left-hand sides of the two reductions to be matched and V is defined over all variables x and terms $t, t_1, \ldots, t_n$ as

$$\begin{aligned}
V(0) &= 2, \\
V(x) &= 2, \\
V(-t) &= 5 \times V(t), \\
V(t_1 + \cdots + t_n) &= V(t_1) + \cdots + V(t_n), \\
V(t_1 \times \cdots \times t_n) &= V(t_1) \times \cdots \times V(t_n).
\end{aligned}$$

This is a natural evaluation function for ring theory problems because the value of a ring sum is simply the integer sum of the values and the value of a ring product is simply the integer product of the values. The value 2 used for constants and variables is the smallest positive integer that is not the additive or multiplicative identity.

The only part of the definition of V that seems contrived for the purpose of the $x^3 = x$ ring problem is the definition of $V(-t)$ as $5 \times V(t)$. This value reflects the fact that $-t = 5t$ is a consequence of of $x^3 = x$ — a fact discovered in the earlier attempt to solve the problem. The choice of $V(-t) = 5 \times V(t)$ as opposed to other reasonable definitions for V is inconsequential because the reduction $-x \rightarrow 5x$ is discovered quite early in the completion process. All other occurrences of $-$ are then eliminated, and the evaluation function for negated terms plays no further role.

11.4. The Proof

The appendix lists the proof of ring commutativity from $x^3 = x$. The proof has been cleaned up slightly and unused inferences are not shown. The program did not use exponentiation or multiplication by a constant, so $3xv^2$ is our shorthand for the program's $(x \times v \times v) + (x \times v \times v) + (x \times v \times v)$.

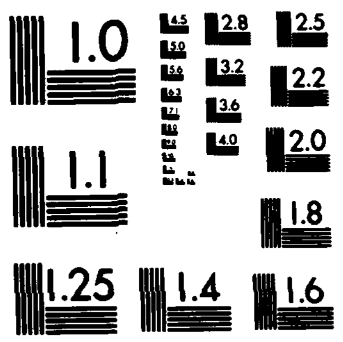
AD-A137 436 RESEARCH ON INTERACTIVE ACQUISITION AND USE OF KNOWLEDGE(U) SRI INTERNATIONAL MENLO PARK CA ARTIFICIAL INTELLIGENCE CENTER B J GROSZ ET AL. NOV 83 3/3

UNCLASSIFIED N00039-80-C-0575 F/G 9/2 NL

END

FILMED

DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Ring axioms were provided as reductions 1-11. These reductions, plus the assumptions of associativity and commutativity for $+$ and associativity for $\times$, constitute a complete set of reductions for free rings. We did not allow matching pairs of ring axiom reductions because this is a complete set of reductions and no new reductions could be created. Reduction (12), not shown, invoked the cancellation laws allowing the derivation of $x = y$ from $x + z = y + z$ and $x = 0$ from $x + z = z$ or $z = x + z$. Reduction (13) is the hypothesis that $x^3 = x$.

Each of the intermediate steps in the derivation is a derived reduction. In effect, reductions are created by forming an expression to which both parent reductions are applicable. The results of applying the two reductions are set equal and fully simplified. If the result is not an identity, it is saved as a new reduction. After each reduction, there may be a line *simplifying x by y* (when there is not, it means the reduction is exactly an equation formed from matching the parent reduction left-hand sides) where x is the equation formed from the two parent reductions and y is the list of simplifiers used to simplify it (*cancel* and *distrib* refer to use of the cancellation and distributivity laws).

A useful perspective is to consider, for example, the derivation of (14) as the simplification of $x^3 = x$ with x instantiated by $y + z$ (i.e., $(y + z)^3 = y + z$) by applying distributivity to $(y + z)^3$.

At the completion of the proof, only 135 reductions had been created of which 53 were retained. The economy of the Knuth-Bendix procedure in number of retained results is demonstrable from the fact that these 135 reductions were the result of simplifying 9,013 equations derived from matching 988 pairs of reductions. Most of the remaining equations were simplified to identities and discarded; a few were simplified to equalities like $3xy = 3yx$ that could not be converted into reductions and were also discarded. Total time was about 14.3 hours (including garbage-collection time) on a Symbolics LM-2 LISP Machine. This reflects slowness of the simplification procedure on numerous lengthy terms and could be greatly reduced.

After the discovery that multiplication is commutative, the problem can be run again to really derive a complete set of reductions for rings with $x^3 = x$. Assuming the associativity and commutativity of addition and multiplication, reductions 1, 2, 3, 5, 6, 7, 9, and 10 comprise a complete set of reductions for free commutative rings. Attempting to complete the set of reductions consisting of these reductions plus $x \times x \times x \rightarrow x$ resulted in the discovery that the reductions marked by ● in the proof comprise a complete set of reductions for rings with $x^3 = x$. During this computation, 120 pairs of reductions were matched and 2121 equations simplified; this resulted in 18 reductions, 8 of which were retained to form the complete set of reductions. The cancellation laws were necessary for deriving this result as well.

The only previous computer proof of the $x^3 = x$ problem was done by Robert Veroff in 1981 using the Argonne National Laboratory - Northern Illinois University theorem-proving program [129]. His solution required an impressive 2^+ minutes on an IBM 3033. It is interesting to compare the approaches taken in these two proofs. Both rely heavily on equality reasoning. The process of fully simplifying equations with respect to a set of reductions is just demodulation. The Knuth-Bendix method's means for deriving equations from pairs of reductions is similar to the paramodulation operation used in the ANL-NIU prover. Cancellation laws were also used by the ANL-NIU prover. Despite such similarities in approach, solution by the Knuth-Bendix method required less preparation of the problem. The Knuth-Bendix program was given only the 11 reductions for free rings, the reduction $x \times x \times x \rightarrow x$, declarations of associativity and commutativity, and a reduction for the cancellation laws. The ANL-NIU program was provided with a total of 60^+ clauses, including the negation of the theorem (their proof was goal-directed, with proving the theorem being a specific objective, whereas our program derived ring commutativity as a result of pure forward reasoning, attempting to complete a set of reductions). Some clauses expressed information about associativity and commutativity, which are handled by declarations in the Knuth-Bendix program. A large number were present to support a polynomial subtraction inference operation—e.g., to derive $a + (-c) = 0$ from $a + b = 0$ and $b + c = 0$. Comparable operations are implicit in the Knuth-Bendix method, which can infer $a = c$ by matching the reductions $x + a + b \rightarrow x$ (the embedded form of $a + b \rightarrow 0$) and $y + b + c \rightarrow y$ (the embedded form of $b + c \rightarrow 0$).

Appendix. Proof that $x^3 = x$ Implies Ring Commutativity

•(1)	$0 + x \rightarrow x$,	
(2)	$(-x) + x \rightarrow 0$,	
•(3)	$x(y + z) \rightarrow xy + xz$,	
(4)	$(x + y)z \rightarrow xz + yz$,	
(5)	$-0 \rightarrow 0$,	
(6)	$-(-x) \rightarrow x$,	
•(7)	$x0 \rightarrow 0$,	
(8)	$0x \rightarrow 0$,	
(9)	$-(x + y) \rightarrow (-x) + (-y)$,	
(10)	$x(-y) \rightarrow -(xy)$,	
(11)	$(-x)y \rightarrow -(xy)$,	
•(13)	$x^3 \rightarrow x$,	
(14)	$zyz + yz^2 + y^2z + z^2y + zy^2 + yzy \rightarrow 0$	from (3) and (13),
	simplifying $(y + z)^2y + (y + z)^2z = y + z$ by cancel, distrib, (13),	
(18)	$yzw + zyw + wyz + wzy + ywz + zwy \rightarrow 0$	from (3) and (14),
	simplifying $w(y + z)w + (y + z)w^2 + (y + z)^2w +$ $w(y + z)^2 + (y + z)w(y + z) + w^2y + w^2z = 0$ by distrib, (14),	
•(21)	$6z \rightarrow 0$	from (13) and (14),
	simplifying $5z^3 + z = 0$ by (13),	
(22)	$3z^2 + 3z \rightarrow 0$	from (13) and (14),
	simplifying $3z^9 + 2z^6 + z^2 = 0$ by (13),	
(24)	$3yz + 3zy \rightarrow 0$	from (3) and (22),
	simplifying $2(y + z)^2 + (y + z)y + (y + z)z + 3y + 3z = 0$ by distrib, (22),	
•(28)	$-r \rightarrow 5r$	from (2)^e and (21)^e,
•(29)	$3v^2 \rightarrow 3v$	from (21)^e and (22)^e,
•(30)	$3xv^2 \rightarrow 3xv$	from (3) and (29),
	simplifying $x(3v) = xv^2 + x(2v^2)$ by distrib,	
(31)	$3v^2z \rightarrow 3vz$	from (4) and (29),
	simplifying $(3v)z = (2v^2)z + v^2z$ by distrib,	
(33)	$3xux + 3ux \rightarrow 0$	from (13)^{e1} and (24),
	simplifying $3x^2ux + 2ux^3 + ux = 0$ by (13), (31),	
(40)	$3xux \rightarrow 3ux$	from (21)^e and (33)^e,
(48)	$yzy + zy^2 + y^2z + 3yz \rightarrow 0$	from (18)^e and (21),
	simplifying $0 = 4yzy + 4zy^2 + 4y^2z$ by (21), (30), (31), (40),	
(60)	$y^2t + ty^2 + yty \rightarrow 3yt$	from (21)^e and (48)^e,
(66)	$2yty + 2y^2t \rightarrow 3yt + ty^2$	from (21)^e and (48)^e,
	simplifying $3yt + ty^2 = 5yty + 5y^2t$ by (24), (31), (40),	
(80)	$xux^2 + x^2ux \rightarrow 2ux$	from (13)^{e1} and (60),
	simplifying $x^2ux + ux + xux^2 = 3xux$ by cancel, (40),	
(82)	$y^2uy^2 + uy^2 + yuy \rightarrow 3uy$	from (13)^{e1} and (60),
	simplifying $y^2uy^4 + uy^2 + yuy^5 = 3yuy^4$ by (13), (30), (40),	
(115)	$2y^2sy \rightarrow ysy^2 + ys$	from (13)^{e2} and (66),
	simplifying $2y^2sy + y^3s + ys = 3y^2s + ysy^2$ by cancel, (13), (31),	
(118)	$y^2sy^2 \rightarrow sy^2$	from (13)^{e2} and (66),
	simplifying $2y^5sy + y^6s + y^2s = 3y^6s + y^4sy^2$ by cancel, (13), (66),	
(119)	$2uy^2 + yuy \rightarrow 3uy$	from (82),
	simplifying it by (118),	
(133)	$xux^2 \rightarrow ux$	from (13)^{e1} and (119),
	simplifying $ux^3 + ux + xux^2 = 3ux^2$ by cancel, (13), (30),	

(135) $x^2ux \rightarrow ux$ from (80),
simplifying it by cancel, (133),

(***) $sy = ys$ from (115),
simplifying it by cancel, (133), (135).

References

[1] Aho, A.V. and S.C. Johnson. LR parsing. *Computing Surveys 6,* 2 (1974), 99-124.

[2] Allen, J.F. and C.R. Perrault. Analyzing intention in utterances. *Artificial Intelligence 15,* 3 (December 1980), 143-178.

[3] Andrews, P.B. Theorem proving via general matings. *Journal of the ACM 28,* 2 (April 1981), 193-214.

[4] Appelt, D.E. Planning natural-language utterances to satisfy multiple goals. Technical Note 259, Artificial Intelligence Center, SRI International, Menlo Park, California, March 1982.

[5] Barwise, J. and J. Perry. *Situations and Attitudes.* 1983.

[6] Bear, J. and L. Karttunen. PSG: a simple phrase structure parser. *Texas Linguistic Forum 14* (1979).

[7] Bibel, W. On matrices with connections. *Journal of the ACM 28,* 4 (October 1981), 633-645.

[8] Bläsius, K., N. Eisinger, J. Siekmann, G. Smolka, A. Herold, and C. Walther. The Markgraf Karl refutation procedure (Fall 1981). *Proceedings of Seventh International Joint Conference on Artificial Intelligence,* Vancouver, British Columbia, August 1981, 511-518.

[9] Bledsoe, W.W. Non-resolution theorem proving. *Artificial Intelligence 9, 1* (August 1977), 1-35.

[10] Bledsoe, W.W. and L.M. Hines. Variable elimination and chaining in a resolution-based prover for inequalities. *Proceedings of Fifth Conference on Automated Deduction,* Les Arcs, France, July 1980, 70-87.

[11] Boyer, R.S. and J S. Moore. The sharing of structure in theorem-proving programs. In B. Meltzer and D. Michie (eds.). *Machine Intelligence 7.* Edinburgh University Press, Edinburgh, Scotland, 1972, pp. 101-116.

[12] Brachman, R.J., R.E. Fikes, and H.J. Levesque. KRYPTON: a functional approach to knowledge representation. *IEEE Computer 16,* 10 (October 1983), 67-73.

[13] Brachman, R.J. and H.J. Levesque. Competence in knowledge representation. *Proceedings of AAAI-82 National Conference on Artificial Intelligence,* Pittsburgh, Pennsylvania, August 1982, 189-192.

[14] Brachman, R.J. and B.C. Smith (eds.). Special Issue on Knowledge Representation. *SIGART Newsletter 70* (February 1980).

[15] Bruce, B. Belief systems and language understanding. Technical Report 21, BBN, Cambridge, Massachusetts, 1975.

[16] Carberry, S. Tracking user goals in an information-seeking environment. *Proceedings of AAAI-83 National Conference on Artificial Intelligence,* Washington, D.C., August 1983.

[17] Chang, C.L. and R.C.T. Lee. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, New York, New York, 1973.

[18] Cohen, P.R. and C.R. Perrault. Elements of a plan-based theory of speech acts. *Cognitive Science 3,* 3 (July-September 1979), 177-212.

[19] Cooper, R. *Quantification and Syntactic Theory.* D.Reidel Publishing Co., Dordrecht, Holland. Forthcoming.

[20] de Kleer, J., J. Doyle, G.L. Steele, and G.J. Sussman. Explicit control of reasoning. In P.H. Winston and R.H. Brown (eds.). *Artificial Intelligence: An MIT Perspective, Vol. 1.* MIT Press, Cambridge, Massachusetts, 1979, 93-116.

[21] Dixon, J.K. Z-resolution: theorem-proving with compiled axioms. *Journal of the ACM 20,* 1 (January 1973), 127-147.

[22] Eder, G. A PROLOG-like interpreter for non-Horn clauses. D.A.I. Research Report No. 26, Department of Artificial Intelligence, University of Edinburgh, Edinburgh, Scotland, September 1976.

[23] Eisinger, N. Subsumption and connection graphs. *Proceedings of Seventh International Joint Conference on Artificial Intelligence,* Vancouver, British Columbia, August 1981, 480-486.

[24] Fikes, R.E. and N.J. Nilsson. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence 2,* 3-4 (Winter 1971), 189-208.

[25] Finin, T. and J. Shraeger. An expert system that volunteers advice. *Proceedings of AAAI-82 National Conference on Artificial Intelligence,* Pittsburgh, Pennsylvania, August 1982, 339-340.

[26] Fleisig, S., D. Loveland, A.K. Smiley III, and D.L. Yarmush. An implementation of the model elimination proof procedure. *Journal of the ACM 21,* 1 (January 1974), 124-139.

[27] Ford, M., J. Bresnan, and R. Kaplan. A competence-based theory of syntactic closure. In J. Bresnan (ed.). *The Mental Representation of Grammatical Relations.* MIT Press, Cambridge, Massachusetts, 1982.

[28] Frazier, L. and J.D. Fodor. The sausage machine: a new two-stage parsing model. *Cognition 6* (1978), 291-325.

[29] Frazier, L. and J.D. Fodor. Is the human sentence parsing mechanism an ATN? *Cognition 8* (1980), 411-459.

[30] Gazdar, G. Phrase structure grammar. July 1980. To appear in P. Jacobson and G.K. Pullum (eds.). *On the Nature of Syntactic Representation.*

[31] Gazdar, G. Unbounded dependencies and coordinate structure, *Linguistic Inquiry 12,* (1981), 165-179.

[32] Genesereth, M.R. The role of plans in automated consultation. *Proceedings of Sixth International Joint Conference on Artificial Intelligence,* Tokyo, Japan, August 1979, 311-319.

[33] Gillogly, J.J. The technology chess program. *Artificial Intelligence 3,* 3 (Fall 1972), 145-163.

[34] Gordon, D. and G. Lakoff. Conversational postulates. In P. Cole and J.L. Morgan (eds.). *Syntax and Semantics, Volume 3: Speech Acts.* Academic Press, New York, New York, 1975.

[35] Gorry G.A., H. Silverman, and S.G. Pauker. Capturing clinical expertise: a computer program that considers clinical responses to digitalis. *American Journal of Medicine 64,* (1978), 452-460.

[36] Grosz, B., N. Haas, G. Hendrix, J. Hobbs, P. Martin, B. Moore, J. Robinson, and S. Rosenschein. DIALOGIC: a core natural-language processing system. *Proceedings of Ninth International Conference on Computational Linguistics,* Prague, Czechoslovakia, July 1982, 95-100.

[37] Haas, N. and G.G. Hendrix. An approach to acquiring and applying knowledge. *Proceedings of AAAI-80 National Conference on Artificial Intelligence,* Stanford, California, 1980, 235-239.

[38] Harrison, M.C. and N. Rubin. Another generalization of resolution. *Journal of the ACM 25,* 3 (July 1978), 341-351.

[39] Hart, P. and R. Duda. PROSPECTOR—a computer-based consultation system for mineral exploration. Technical Note 155, Artificial Intelligence Center, SRI International, Menlo Park, California, October 1977.

[40] Hayes, P.J. Computation and deduction. *Proceedings of Second Symposium on Mathematical Foundations of Computer Science,* Czechoslovak Academy of Sciences, September 1973, 105-116.

[41] Hayes, P.J. In defence of logic. *Proceedings of Fifth International Joint Conference on Artificial Intelligence,* Cambridge, Massachusetts, August 1977, 559-565.

[42] Hendrix, G.G. The LIFER manual: a guide to building practical natural language interfaces. Technical Note 138, Artificial Intelligence Center, SRI International, Menlo Park, California, February 1977.

[43] Hewitt, C. Description and theoretical analysis using schemata of PLANNER: a language for proving theorems and manipulating models in a robot. Technical Report AI TR-258, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, April 1972.

[44] Hewitt, C., P. Bishop, and R. Steiger. A universal modular ACTOR formalism for artificial intelligence. *Advance Papers of the Third International Conference on Artificial Intelligence,* Stanford, California, August 1973, 235-245.

[45] Hewitt, C. How to use what you know. *Advance Papers of the Fourth International Joint Conference on Artificial Intelligence,* Tbilisi, Georgia, USSR, September 1975, 189-198.

[46] Hirschberg, J. *Scalar and Hierarchical Inferencing in Question/Answer Exchanges.* Doctoral disseration proposal, University of Pennsylvania, Philadelphia, Pennsylvania, 1983.

[47] Hobbs, J.R. and J.J. Robinson Why ask?. Technical Note 169, Artificial Intelligence Center, SRI International, Menlo Park, California, October 1982.

[48] Hsiang, J. Refutational theorem proving using term rewriting systems. Unpublished manuscript, Department of Computer Science, University of Illinois, Urbana, Illinois, July 1981.

[49] Huet, G. Confluent reductions: abstract properties and applications to term rewriting systems. *Journal of the ACM 27,* 4 (October 1980), 797-821.

[50] Huet, G. A complete proof of the correctness of the Knuth-Bendix completion algorithm. *Journal of Computer and System Sciences 23,* 1 (August 1981), 11-21.

[51] Hughes, G.E. and M.J. Cresswell. *An Introduction to Modal Logic.* Methuen and Company Ltd., London, England, 1968.

[52] Hullot, J.M. A catalogue of canonical term rewriting systems. Technical Report CSL-113, Computer Science Laboratory, SRI International, Menlo Park, California, April 1980.

[53] Kaplan, S.J. *Cooperative Responses from a Portable Natural Language Data Base Query System.* Ph.D. Disseration, University of Pennsylvania, Philadelphia, Pennsylvania, 1979.

[54] Kaplan, R.M. and M. Kay. Personal communication. 1981.

[55] Kaplan, R.M. and M. Kay. Phonological rules and finite state transducers. Paper presented at the 56th Annual Meeting of the LSA, New York, New York,

[56] Karttunen, L. KIMMO: a general morphological processor. *Texas Linguistic Forum 22* (1983), 161-185. December 1981.

[57] Karttunen, L., R. Root, and H. Uszkoreit. Morphological analysis of Finnish by computer. Paper presented at the ACL session of the 54th Annual Meeting of the LSA, New York, New York, December 1981.

[58] Kautz, H.A. A first-order dynamic logic for planning. Technical Report CSRG-144, University of Toronto, Toronto, Canada, 1982.

[59] Kimball, J. Seven principles of surface structure parsing in natural language. *Cognition 2,* 1 (1973), 15-47.

[60] Knuth, D.E. and P.B. Bendix. Simple word problems in universal algebras. In J. Leech (ed.). *Computational Problems in Abstract Algebras,* Pergamon Press, 1970, pp. 263-297.

[61] Konolige, K. Capturing linguistic generalizations with metarules in an annotated phrase structure grammar. *Proceedings of 18th Annual Meeting of the Association for Computational Linguistics,* University of Pennsylvania, Philadelphia, Pennsylvania, June 1980.

[62] Konolige, K. *A Deduction Model of Belief.* Ph.D. Dissertation in preparation, Stanford University, Stanford, California, 1983.

[63] Kornfeld, W.A. Equality for Prolog. *Proceedings of Eighth International Joint Conference on Artificial Intelligence,* Karlsruhe, West Germany, August 1983.

[64] Koskenniemi, K. *A Two-level Model for Morphological Analysis and Synthesis.* Forthcoming Ph.D. dissertation, University of Helsinki, Helsinki, Finland.

[65] Kowalski, R. Predicate logic as a programming language. *Information Processing 74.* North-Holland Publishing Company, Amsterdam, The Netherlands, 1974, pp. 569-574.

[66] Kowalski, R. A proof procedure using connection graphs. *Journal of the ACM 22,* 4 (October 1975), 572–595.

[67] Kowalski, R. *Logic for Problem Solving.* Elsevier North Holland, Inc., New York, New York, 1979.

[68] Kowalski, R. and D. Kuehner. Linear resolution with selection function. *Artificial Intelligence 2* (1971), 227–260.

[69] Lankford, D.S. Canonical algebraic simplification in computational logic. Technical Report, Department of Mathematics, University of Texas, Austin, Texas, May 1975.

[70] Lankford, D.S. Canonical inference. Report ATP-32, Department of Mathematics, University of Texas, Austin, Texas, December 1975.

[71] Lankford, D.S. and A.M. Ballantyne. Decision procedures for simple equational theories with commutative axioms: complete sets of commutative reductions. Report ATP-35, Department of Mathematics, University of Texas, Austin, Texas, March 1977.

[72] Lankford, D.S. and A.M. Ballantyne. Decision procedures for simple equational theories with permutative axioms: complete sets of permutative reductions. Report ATP-37, Department of Mathematics, University of Texas, Austin, Texas, April 1977.

[73] Lankford, D.S. and A.M. Ballantyne. Decision procedures for simple equational theories with commutative-associative axioms: complete sets of commutative-associative reductions. Report ATP-39, Department of Mathematics, University of Texas, Austin, Texas, August 1977.

[74] Litman, D. *Discourse and Problem Solving.* Doctoral dissertation proposal, University of Rochester, Rochester, New York, 1983.

[75] Livesey, M. and J. Siekmann. Termination and decidability results for string-unification. Memo CSM-12, Essex University Computing Center, Colchester, Essex, England, August 1975.

[76] Livesey, M. and J. Siekmann. Unification of A+C-terms (bags) and A+C+I-terms (sets). Interner Bericht Nr. 5/76, Institut für Informatik I, Universität Karlsruhe, Karlsruhe, West Germany, 1976.

[77] Loveland, D.W. A simplified format for the model elimination procedure. *Journal of the ACM 16,* 3 (April 1969), 349–363.

[78] Loveland, D.W. *Automated Theorem Proving: A Logical Basis.* North-Holland, Amsterdam, The Netherlands, 1978.

[79] Loveland, D.W. and M.E. Stickel. The hole in goal trees: some guidance from resolution theory. *Proceedings of Third International Joint Conference on Artificial Intelligence,* Stanford, California, August 1973, 153–161. Reproduced in *IEEE Transactions on Computers C-25,* 4 (April 1976), 335–341.

[80] *The MACSYMA Reference Manual.* Massachusetts Institute of Technology, Cambridge, Massachusetts, 1974.

[81] Manna, Z. and R. Waldinger. A deductive approach to program synthesis. *ACM Transactions on Programming Languages and Systems 2,* 1 (January 1980), 90–121.

[82] Manna, Z. and R. Waldinger. Special tions in program-synthetic deduction (a summary). To appear in *Journal of the AC*

[83] Marcus, M. *A Theory of Syntactic Recognition for Natural Language.* MIT Press, Cambridge, Massachusetts, 1980.

[84] Mays, E. Correcting misconceptions about data base structure. *Proceedings of Conference of Canadian Society for Computational Studies of Intelligence.* 1980.

[85] McCarthy, J. Circumscription—a form of non-monotonic reasoning. *Artificial Intelligence 13,* 1-2 (April 1980), 27-39.

[86] McDermott, D. Planning and acting. *Cognitive Science 2,* 2 (April-June 1978), 71-109.

[87] Minsky, M. A framework for representing knowledge. Technical Report AIM-306, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, June 1974. Portions reprinted in J. Haugeland (ed.). *Mind Design: Philosophy, Psychology, and Artificial Intelligence.* Bradford Books, Publishers, Inc., Montgomery, Vermont, 1980.

[88] Montague, R. The proper treatment of quantification in ordinary English. In R.H. Thomason (ed.). *Formal Philosophy, Selected Papers of Richard Montague.* Yale University Press, New Haven, Connecticut, 1974, pp. 188-221.

[89] Moore, R.C. Reasoning from incomplete knowledge in a procedural deduction system. Technical Report AI-TR-437, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts (December 1975). Also published by Garland Publishing, Inc., New York, New York, 1980.

[90] Moore, R.C. Reasoning about knowledge and action. Technical Note 191, Artificial Intelligence Center, SRI International, October 1980.

[91] Moore, R.C. Problems in logical form. *Proceedings of 19th Annual Meeting of the Association for Computational Linguistics,* Stanford, June 1981.

[92] Morris, J.B. E-resolution: extension of resolution to include the equality relation. *Proceedings of International Joint Conference on Artificial Intelligence,* Washington, D.C., May 1969, 287-294.

[93] Murray, N.V. Completely non-clausal theorem proving. *Artificial Intelligence 18,* 1 (January 1982), 67-85.

[94] Nelson, G. and D.C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems 1,* 2 (October 1979), 245-257.

[95] Newell, A. The knowledge level. *AI Magazine 2,* 2 (Summer 1981), 1-20.

[96] Pereira, F.C.N. A new characterisation of attachment preferences. To appear in D. Dowty, L. Karttunen, and A. Zwicky (eds.). *Natural Language Processing. Psycholinguistic, Computational, and Theoretical Perspectives.* Cambridge University Press, Cambridge, England.

[97] Pereira, F.C.N. and S.M. Shieber. Shift-reduce scheduling and syntactic closure. To appear.

[98] Pereira, F.C.N. and D.H.D. Warren. Parsing as deduction. *Proceedings of 21st Annual Meeting of the Association for Computational Linguistics,* Cambridge, Massachusetts, June 1983.

[99] Perrault, C.R. and J.F. Allen. A plan-based analysis of indirect speech acts. *American Journal of Computational Linguistics 6,* 3 (1980), 167-182.

[100]Peters, S. and R.W. Ritchie. Phrase linking grammars. Unpublished manuscript, December 1981.

[101]Peterson, G.E. and M.E. Stickel. Complete sets of reductions for some equational theories. *Journal of the ACM 28,* 2 (April 1981), 233-264.

[102]Peterson, G.E. and M.E. Stickel. Complete systems of reductions using associative and/or commutative unification. Technical Note 269, Artificial Intelligence Center, SRI International, Menlo Park, California, October 1982. To appear in M. Richter (ed.). *Lecture Notes on Systems of Reductions.*

[103]Plotkin, G.D. Building-in equational theories. In Meltzer, B. and D. Michie (eds.). *Machine Intelligence 7.* Edinburgh University Press, Edinburgh, Scotland, 1972, pp. 73-90.

[104]Pollack, M., J. Hirschberg, and B. Webber. User participation in the reasoning processes of expert systems. *Proceedings of AAAI-82 National Conference on Artifical Intelligence,* Pittsburgh, Pennsylvania, August 1982, 358-361. A longer version appears as University of Pennsylvania Technical Report MS-CIS-82-9.

[105]Pratt, V.R. Six lectures on dynamic logic. Technical Report MIT/LCS/TM-117, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1978.

[106]Raulefs, P., J. Siekmann, P. Szabo, and E. Unvericht. A short survey on the state of the art in matching and unification problems. *SIGSAM Bulletin 13,* 2 (May 1979), 14-20.

[107]Reynolds, J. Unpublished seminar notes. Stanford University, Palo Alto, California, Fall 1965.

[108]Rich, C. Knowledge representation languages and predicate calculus: how to have your cake and eat it too. *Proceedings of AAAI-82 National Conference on Artificial Intelligence,* Pittsburgh, Pennsylvania, August 1982, 193-196.

[109]Robinson, J.A. A machine-oriented logic based on the resolution principle. *Journal of the ACM 12,* 1 (January 1965), 23-41.

[110]Robinson, J.J. DIAGRAM: a grammar for dialogues. *Communications of the ACM 25,* 1 (January 1982), 27-47.

[111]Rosenschein, S.J. Plan synthesis: a logical perspective. *Proceedings of Eighth International Joint Conference on Artificial Intelligence.* Vancouver, British Columbia, Canada, August 1981, 331-337.

[112]Rulifson, J.F., J.A. Derksen, and R.J. Waldinger. QA4: a procedural calculus for intuitive reasoning. Technical Note 73, Artificial Intelligence Center, SRI International, Menlo Park, California, November 1972.

[113]Sacerdoti, E. *A Structure for Plans and Behavior.* American Elsevier, New York, New York, 1977.

[114]Schmidt, C.F., N.S. Sridharan, and J.L. Goodson. The plan recognition problem: an intersection of artificial intelligence and psychology. *Artificial Intelligence 11,* 1-2 (August 1978), 45-83.

[115]Searle, J. Indirect speech acts. In P. Cole and J.L. Morgan (eds.). *Syntax and Semantics, Volume 3: Speech Acts.* Academic Press, New York, New York, 1975.

[116]Shieber, S. Notes on first-order dynamic logic and bigression. Unpublished notes, 1981.

[117]Shieber, S. Direct parsing of ID/LP grammars. Technical Note 291, Artificial Intelligence Center, SRI International, Menlo Park, California, August 1983.

[188]Shostak, R.E. Refutation graphs. *Artificial Intelligence 7,* 1 (Spring 1976), 51-64.

[119]Shostak, R.E. Deciding combinations of theories. *Proceedings of Sixth Conference on Automated Deduction,* New York, New York, June 1982, 209-222.

[120]Sickel, S. A search technique for clause interconnectivity graphs. *IEEE Transactions on Computers C-25,* 8 (August 1976), 823-835.

[121]Siekmann, J. String-unification, part I. Essex University, Cochester, Essex, England. March 1975.

[122]Siekmann, J. T-unification, part I. Unification of commutative terms. Interner Bericht Nr. 4/76, Institut für Informatik I, Universität Karlsruhe, Karlsruhe, West Germany, 1976.

[123]Slagle, J.R. Automatic theorem proving with renamable and semantic resolution. *Journal of the ACM 14,* 4 (October 1967), 687-697.

[124]Slagle, J.R. and L.M. Norton. Experiments with an automatic theorem-prover having partial ordering inference rules. *Communications of the ACM 16,* 11 (November 1973), 682-688.

[125]Stickel, M.E. The programmable strategy theorem prover: an implementation of the linear MESON procedure. Technical Report, Carnegie-Mellon University Computer Science Department, Pittsburgh, Pennsylvania, June 1974.

[126]Stickel, M.E. Mechanical theorem proving and artificial intelligence languages. Ph.D. Dissertation, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania, December 1977.

[127]Stickel, M.E. A unification algorithm for associative-commutative functions. *Journal of the ACM 28,* 3 (July 1981), 423-434.

[128]Tyson, W.M. *APRVR: A Priority-Ordered Agenda Theorem Prover.* Ph.D. Dissertation, University of Texas at Austin, Austin, Texas, 1981.

[129]Veroff, R.L. Canonicalization and demodulation. Report ANL-81-6, Argonne National Laboratory, Argonne, Illinois, February 1981.

[130]Wanner, E. The ATN and the sausage machine: which one is baloney? *Cognition 8* (1980), 209-225.

[131]Warren, D.H.D. and L.M. Pereira. PROLOG—the language and its implementation compared with LISP. *Proceedings of Symposium on Artificial Intelligence and Programming Languages. SIGPLAN Notices 12,* 8 and *SIGART Newsletter 64* (August 1977), 109-115.

[132]Winker, S.K. and L. Wos. Procedure implementation through demodulation and related tricks. *Proceedings of Sixth Conference on Automated Deduction,* New York, New York, June 1982, 109–131.

[133]Winograd, T. *Understanding Natural Language.* Academic Press, New York, New York, 1972.

[134]Wos, L. and G.A. Robinson. Paramodulation and set of support. *Proceedings of Symposium on Automatic Demonstration,* Versailles, France, 1968, *Lecture Notes in Mathematics 125,* Springer-Verlag, Berlin, West Germany (1970), 276–310.

[135]Wos, L., G. Robinson, D. Carson, and L. Shalla. The concept of demodulation in theorem proving. *Journal of the ACM 14,* 4 (October 1967), 698–709.